Naval Research Laboratory
Stennis Space Center, MS 39529-5004

MDFF HELP Library, Version 2: On-Line Documentation for the Map Data Formatting Facility

S. A. MYRICK
P. B. WISCHOW
M. C. LOHRENZ
M. E. TRENCHARD

Mapping, Charting, and Geodesy Branch
Marine Geosciences Division

M. L. GENDRON
L. M. RIEDLINGER
J. M. MEHAFFEY

Planning Systems, Inc.
Slidell, LA 70458

March 7, 1994

REPORT DOCUMENTATION PAGE

1. Agency Use Only *(Leave blank)*.	2. Report Date. March 7, 1994	3. Report Type and Dates Covered. Final

4. Title and Subtitle.

MDFF HELP Library, Version 2: On-Line Documentation
for the Map Data Formatting Facility

5. Funding Numbers.

Program Element No. 0604214N

Project No.

6. Author(s).

S. A. Myrick, P. W. Wischow, M. C. Lohrenz, M. E. Trenchard, M. L. Gendron*,
L. M. Riedlinger*, and J. M. Mehaffey*

Task No.

Accession No. DN257017

Work Unit No. 74526903

7. Performing Organization Name(s) and Address(es).

Naval Research Laboratory
Marine Geosciences Division
Stennis Space Center, MS 39529-5004

8. Performing Organization Report Number.

NRL/MR/7441--93-7071

9. Sponsoring/Monitoring Agency Name(s) and Address(es).

Naval Research Laboratory
Naval Air Systems Command
Washington, DC 20361-8030

10. Sponsoring/Monitoring Agency Report Number.

NRL/MR/7441--93-7071

11. Supplementary Notes.

12a. Distribution/Availability Statement.

12b. Distribution Code.

13. Abstract *(Maximum 200 words)*.

The purpose of this report is to describe on-line documentation for the Naval Research Laboratory's Map Data Formatting Facility (MDFF). The MDFF utilizes Digital Equipment Corporation (DEC) computers to process several types of data including Compressed Aeronautical Chart, Compressed Nautical Chart, and Digital Landmass System. Using DEC software utilities, on-line documentation has been developed that provides information pertaining to the processing and compression of these data types and to other topics that are specific to the MDFF.

14. Subject Terms.

Digital Maps, Optical Storage, Databases, Data Compression

15. Number of Pages.

207

16. Price Code.

17. Security Classification of Report. Unclassified	18. Security Classification of Report. Unclassified	19. Security Classification of This Page. Unclassified	20. Limitation of Abstract. of Abstract. SAR

Contents

MDFF HELP Library Version 2:
On-line Documentation for the Map Data Formatting Facility

INTRODUCTION

The Map Data Formatting Facility (MDFF) utilizes Digital Equipment Corporation (DEC) computer hardware and software in its research and development efforts. This memorandum report assumes that the reader is familiar with the VAX/VMS operating system. A VAX/VMS utility is used to create on-line documentation in the form of a HELP library, which is referred to as the MDFF HELP library because it contains topics that are specific to the MDFF. The intent of the MDFF HELP library is to provide accurate and current information that is readily available, user friendly, and easily maintained. Because the MDFF is a research and development project, continual updates to the MDFF HELP library will be required.

FUNCTIONALITY/REQUIREMENTS

The presentation and recovery of on-line documentation within the MDFF HELP library is functionally similar to the VAX/VMS HELP library. The MDFF HELP library is invoked by using the symbol DEC Control Language (DCL) symbol **MDFFHELP**. **MDFFHELP** is a convenient symbol for simplifying the day-to-day usage of the rather lengthy VAX/VMS librarian command:

HELP/LIBRARY = MDFF_SYSTEM:[DOCS_HELP]MDFF.HLB

Once the MDFF HELP utility is invoked, alphabetized topics are displayed. Information about a specific topic is made available by entering the name of a topic at the prompt. Main topics currently provided by the MDFF HELP utility are listed in Figure 1. Note that several types of data are processed by the MDFF, including Compressed Aeronautical Chart (CAC), Compressed Nautical Chart (CNC), and Digital Landmass System (DLMS).

Topics pertaining to specific data types are differentiated by using the data type as part of the topic name. For example, the topic named CAC_PROCESSING provides documentation relevant to CAC data processing. Likewise, the CNC_PROCESSING topic provides documentation relevant to CNC data processing. Topic names, though lengthy, are descriptive and follow DEC recommendations for restricting names to upper and lower cases, digits, underscores, and hyphens.

The VAX/VMS library utility is used to create the MDFF HELP library. Individual text files, which contain information pertaining to individual topics, are written according to a specified format and are directly input to the library. For

1

example, a single text file, which contains information about the topic CAC_PROCESSING, is used to build that portion of the library. By default, text files that are used to build the MDFF HELP library must have the file extension **HLP**.

The VAX/VMS library utility requires the text files to be written in a specific format. Each file must contain key numbers in the first column, followed by the name of the key. Topics that will appear in the initial topic listing, as shown in Figure 1, must use key one. All subtopics use keys two through nine, and consist of a key number followed by the name of the subkey. Figure 2 lists the text file named ARCHIVE.HLP (the first 20 lines) and the format used. The appendices contain listings of current topic files. Appendices A through N contain components of CAC processing and utility software.

CREATION and MAINTENANCE

A DCL command file (Figure 3) is used to invoke the VAX/VMS library utility and to create MDFF HELP. Note that the command file contains two commands: the first command creates the MDFF HELP library, and the second command inserts the topics. Because all topic files have the extension **HLP**, they are automatically inserted into the MDFF HELP library. Presently, this command file is located on the directory MDFF_SYSTEM:[DOCS_HELP] and is named MDFFHELP_REBUILD.COM. The following syntax is used for invocation:

@MDFFHELP_REBUILD.COM

MDFFHELP_REBUILD.COM is used for adding new topics to MDFF HELP and for modifying existing ones. For inclusion into MDFF HELP, all topic files must reside in the directory MDFF_SYSTEM:[DOCS_HELP].

SUMMARY

On-line documentation for the MDFF is available through the MDFF HELP library. The MDFF HELP library provides information that is readily available, user friendly, and easily maintained. Format requirements for individual topic files are defined. The command file, MDFFHELP_REBUILD.COM, is utilized for proper creation and maintenance.

The MDFF HELP library will require updates in keeping with continuing research and development efforts of the MDFF project.

ACKNOWLEDGMENTS

2

```
HELP

 This HELP LIBRARY is specific to NRL MDFF Laboratory
 topics.  For DEC VAX/VMS topics use the default
 VAX/VMS HELP LIBRARY.



 Additional information available:

 Acronyms     Archive    Bitmaps     CAC_Processing
 CAC_Program_Descriptions  CAC_Source_Code   CNC_Processing
 Definitions             DLMS_Processing        HELP    Hints
 Logical_Names           Mapstation    Processing_Threads
 Symbols
```

Figure 1. MDFF HELP Main Topics

```
1 Archive
  ARCHIVE is a program that is used to display and maintain
  MDFF archive data sets.  MDFF archive data sets serve
  several purposes including:

  * Providing historical research data.
  * Providing examples of significant features.
  * Establishing data sets for base-line testing.
  * Establishing data sets for demonstrations.

2 Overview
  Type the following command to execute ARCHIVE:

      RUN/NODEB MDFFEXE:ARCHIVE

  ARCHIVE is menu driven.  The main menu offers the following
  options (which are described as subtopics).

    VIEW: Displays the archive data set.
    ADD: Adds new data into the ARCHIVE data set.
```

Figure 2. MDFF HELP Library Text File and Format

```
$!*************************************************************
$!                                                            *
$! Command file for building the MDFF HELP Library            *
$!                                                            *
$! Written by:    Stephanie A. Myrick   10/11/91              *
$!                                                            *
$!*************************************************************
$!
$! Recreate MDFF HELP library
$ library/create=(keysize=25)/help mdff_system:[docs_help]mdff
$!
$!
$! Insert all topics - files with HLP extension
$ library/insert/help/log mdff_system:[docs_help]mdff  -
                         mdff_system:[docs_help]*.hlp
```

Figure 3. Command File for Building MDFF HELP

APPENDIX A
TOPIC FILE ACRONYMS.HLP

TOPIC FILE: ACRONYMS.HLP

The following text and subtopics appear when **Acronyms** is selected as an **MDFFHELP** topic:

```
Acronyms
 Acronyms are used throughout the MDFF environment.  This
 topic defines some commonly used acronyms.

Additional information available:

1:2M        1:1M        1:500K      1:250K      1:100K      1:50K
ADRG        AGL         AMSL        AOD         ARC         ASCII
ASWPC       CAC         CD          CDROM       CHUM        CLUT
CN          CNC         CRT         CSVQ        DFAD        DFID
DLMS        DMA         DMAAC       DMS         DR          DTED
ECHUM       EEPROM      EQ          FACS        FIPS
FIPSPUB     FLIPS       FRC         GLCC        GNC         HTI
ISO         JNC         JOG         KNTC        LCC         MC&G
MDFF        MEF         MO          MOMS        MTE         NOTAM
NP          NT          ODI         ONC         PA          PNTC
RGB         SEC         SP          ST          TLM         TPC
TS          VFR         VFRTA       VQ          WGS         WORM
YMC         ZDR
```

The following text comprises this **MDFFHELP** topic file. Note that subtopics begin with the key "2", which is located in column 1.

1 Acronyms
     Acronyms are used throughout the MDFF environment. This topic
     defines some commonly used acronyms.

2 1:2M
        1:2,000,000 scale chart.

2 1:1M
        1:1,000,000 scale chart.

2 1:500K
        1:500,000 scale chart.

2 1:250K
        1:250,000 scale chart.

2 1:100K
       1:100,000 scale chart.

2 1:50K
       1:50,000 scale chart.

2 ADRG
       ARC Digitized Raster Graphics. A 24 bit scanned chart
       database that is produced by the Defense Mapping Agency
       at a resolution of 256 pixels per inch.

2 AGL
       Above Ground Level.

2 AMSL
       Above Mean Sea Level.

2 AOD
       Aircraft Optical Disk. A WORM to which a map-station
       optical disk image is written. The AOD is used by the
       Digital Moving Map System onboard tactical aircraft
       for in-flight navigation.

2 ARC
       Equal Arc-Second Raster Chart. A type of Tessellated
       Spheroid system used to store ADRG data.

2 ASCII
       American Standard Code for Information Interchange.
       A standard for information exchange between
       equipment produced by different manufacturers.

2 ASWPC
       Anti-Submarine Warfare Plotting Charts (1:1,166,614 scale).

2 CAC
       Compressed Aeronautical Chart. A 2-bits per pixel
       version of aeronautical ARC Digitized Raster Graphics,
       that is stored in Tessellated Spheroid projection for
       use in mission planning systems and Digital Moving Map
       Systems. The resolution is 128 pixels per inch.

2 CD
       Compact Disk - Read Only Memory. Optical storage media
       that holds approximately 600 Mbytes.

       Also referred to as CD-ROM.

2 CD-ROM
       Compact Disk - Read Only Memory. Optical storage media
       that holds approximately 600 Mbytes.
       Also referred to as CD.

2 CHUM
>   Chart Updating Manual. A semiannual publication that
>   supplies modifications to published aeronautical charts.
>   Additional information is in the DEFINITIONS topic CHUM.

2 CLUT
>   Color Lookup Table. Used to determine the closest 8-bit
>   color in the color palette for a given 24-bit (red, green,
>   blue) value.

2 CN
>   Color Normalization. A color preserving technique that is
>   used to build the spatial compression codebook. During CN,
>   compression codewords are selected based on the normalized
>   use of each of the codeword's four colors in the color
>   compressed image.

2 CNC
>   Compressed Nautical Chart. A 2-bits per pixel version
>   of nautical ADRG, stored in Tessellated Spheroid projection
>   for use in mission planning systems and Digital Moving
>   Map Systems. CNC format is identical to Compressed
>   Aeronautical Chart format. The resolution is 128 pixels
>   per inch.

2 CRT
>   Cathode Ray Tube. A vacuum tube in which a hot cathode
>   emits electrons that are accelerated as a beam through a
>   relatively high voltage anode, further focused or deflected
>   electrostatically or electromagnetically, and allowed to
>   fall on a fluorescent screen.

2 CSVQ
>   Color and Spatial Vector Quantization. An iterative
>   technique for locating vector centroids that may be used to
>   color compress and then spatially compress an image.

2 DFAD
>   Digital Feature Analysis Data. A Defense Mapping
>   Agency database that consists of cultural features
>   (e.g., roads, metropolitan areas, major landmarks).

2 DFID
>   Digital Flight Information Data. Defense Mapping
>   Agency database.

2 DLMS
>   Digital Land Mass System. A database consisting of Digital
>   Terrain Elevation Data and Digital Feature Analysis
>   Data.

2 DMA
    Defense Mapping Agency. The Defense Department's
    primary source of digital and analog maps and charts.

2 DMAAC
    Defense Mapping Agency Aerospace Center.

2 DMS
    Digital Moving Map System. Computer system on board
    tactical aircraft that displays digital chart data and
    selected overlays for in-flight navigation.

2 DR
    Distribution Rectangle. The minimum bounding rectangle in
    geographic coordinates encompassing a geographic contiguous
    set of ADRG image data (that is, the bounding geographic
    coordinates of the data on the ARC Digitized Raster
    Graphics CD-ROM).

2 DTED
    Digital Terrain Elevation Data. Defense Mapping Agency
    standard elevation data set in which values are placed every
    3 degrees of latitude and longitude.

2 ECHUM
    Electronic Chart Updating Manual.

2 EEPROM
    Electronic Erasable Programmable Read-Only Memory.
    Read-Only memory can be erased and reprogrammed as many
    times as the user desires.

2 EQ
    Equatorial (TS zone 2). The region of the earth bounded by
    an upper latitude of +31.3846 and a lower latitude of
    -31.3846.

2 FACS
    Feature Attribute Coding Standard.

2 FIPS
    Federal Information Processing Standards.

2 FIPSPUB
    FIPS Publication. A publication released by the FIPS
    committee.

2 FLIPS
    Flight Information Publications.

2 FRC
    Fallon Range Chart (1:500K scale).

2 GLCC

Global Loran Navigation and Planning Charts (1:5M scale).

2 GNC

Global Navigation and Planning Charts (1:5M scale).

2 HTI

Horizons Technology, Incorporated. Developer of the Map, Operator, and Maintenance Station (MOMS).

2 ISO

International Standards Organization. An organization formed to define standards. For example, ISO8211 is the standard for transfer of geographic information and ISO9660 is the standard/format for CD-ROM data storage.

2 JNC

Jet Navigation Chart (1:2M scale aeronautical chart).

2 JOG

Joint Operational Graphics (1:250K scale aeronautical chart).

2 KNTC

Korean Navigation Training Chart. This chart is available in two scales:

         1:500K scale
         1:1M scale

2 LCC

Loran-C Navigational Chart (1:3M scale).

2 MC&G

Mapping, Charting, and Geodesy. Topics within geography. Divisional name of NRL, Code 7440.

2 MDFF

Map Data Formatting Facility. A project within NRL, Code 7441, that performs research, development, testing, and evaluation in the fields of digital map data and image compression.

2 MEF

Maximum Elevation Figure. A value indicating the maximum elevation within a certain area.

2 MO

Magneto-Optical Disk. Storage media used in the operator portion of the Map, Operator and Maintenance Station.

2 MOMS

Map, Operator and Maintenance Station. Developed by HTI.

A-7

The Map Station portion is used in the MDFF to produce Aircraft Optical Disks.

The Operator Station portion is used by mission planners and permits addition of overlay data (threats, routes, targets, etc.) to DMS via EEPROM device.

The Maintenance portion is used after flights to evaluate aircraft performance.

2 MTE
Maximum Terrain Elevation. A value indicating the maximum elevation within a certain area.

2 NOTAM
Notice to Airmen.

2 NP
North Polar (TS zone 4). The region of the earth bounded by an upper latitude of +90.0000 and a lower latitude of +51.6923.

2 NT
North Temperate (TS zone 3). The region of the earth bounded by an upper latitude of +51.6923 and a lower latitude of +31.3846.

2 ODI
Optical Disk Image. An image file containing Red, Green, and Blue data (along with related color lookup tables, decompression codebooks, etc.) which resides on codebooks, etc.), which resides on optical disk media (e.g., WORM, MO, CD-ROM).

2 ONC
Operational Navigation Chart (1:1M scale aeronautical chart).

2 PA
Palette. There are 30 standard color palettes in the CAC database. There is one palette for each scale and zone. Each color palette contains 256 colors: 240 colors are used for chart data and the other 16 colors are reserved for geographic overlays.

2 PNTC
Philippines Navigational Training Chart. This chart is available in two scales:

      1:500K scale
       1:1M scale

2 RGB

Red - Green - Blue. A color scheme often used in color display devices (e.g., color monitors and raster displays).

2 SEC

Sectional Aeronautical Charts (1:500K scale).

2 SP

South Polar (TS zone 0). The region of the earth bounded by an upper latitude of -51.6923 and a lower latitude of -90.0000.

2 ST

South Temperate (TS zone 1). The region of the earth bounded by an upper latitude of -31.3846 and a lower latitude of -51.6923.

2 TLM

Topographic Line Map (1:50K and 1:100K scale aeronautical charts).

2 TPC

Tactical Pilotage Chart (1:500K scale aeronautical chart).

2 TS

Tessellated Spheroid. The map projection system in which the Compressed Aeronautical Chart and the Compressed Nautical Chart are stored.

2 VFR

Visual Flight Rules.

2 VFRTA

Visual Flight Rules Terminal Area Charts (1:250 scale).

2 VQ

Vector Quantization. Lossy compression method used by the MDFF to compress ADRG data.

2 WGS

World Geodetic System. A standard by which the earth's geoid (elliptical shape) is measured.

2 WORM

Write Once, Read Many-times. An optical disk used to store ODIs and intended for use in aircraft Digital Moving Map Systems. A WORM can store approximately 240 Mbytes of chart data on each side.

2 YMC

Yellow-Magenta-Cyan. A color scheme often used by hardcopy devices that deposit colored pigments onto paper (e.g.,

CalComp plotters).

2 ZDR

Zone Distribution Rectangle. An ADRG image for a given ARC zone.

APPENDIX B
TOPIC FILE ARCHIVE.HLP

TOPIC FILE: ARCHIVE.HLP

The following text and subtopics appear when **Archive** is selected as
an **MDFFHELP** topic:

Archive

Archive is a program that is used to display and maintain MDFF
archive data sets. MDFF archive data sets serve several
purposes including:

* Providing historical research data.
* Providing examples of significant features.
* Establishing data sets for base-line testing.
* Establishing data sets for demonstrations.

Additional information available:

Overview   VIEW_Datasets   ADD_Datasets
DELETE_Datasets   END

The following text comprises this **MDFFHELP** topic file. Note that
subtopics begin with the key "2", which is located in column 1.

1 Archive
 Archive is a program that is used to display and maintain MDFF
 archive data sets. MDFF archive data sets serve several
 purposes including:

    * Providing historical research data.
    * Providing examples of significant features.
    * Establishing data sets for base-line testing.
    * Establishing data sets for demonstrations.

2 Overview

 Type the following command to execute the program:

        RUN/NODEB MDFFEXE:ARCHIVE

 Archive is menu driven. The main menu offers the following
 options (which are described as subtopics).

    VIEW: Displays the archive data set.
     ADD: Adds new data into the archive data set.

DELETE: Deletes data from the archive data sets.
   END: Ends (terminates) program execution.

An option is selected by using the arrow keys to highlight the desired option and pressing <CR>.

The following restrictions apply to usage:
   * The user must be logged onto a VAXstation running VAX/VWS to VIEW archive data.
   * Any computer may be used to ADD or DELETE archive data sets; however, the user must be logged in under the CAC account.

2 VIEW_Datasets

This main menu option is used to display archive data onto a VAX/VWS workstation color monitor. The user must be logged onto a VAXstation running VAX/VWS software in order to view archive data sets.

* A submenu is displayed for data set selection. Submenu options include CAC, CNC, or DLMS data sets (at present, only CAC is implemented).

   To select a CAC data set use the arrow keys to highlight the CAC option and press <CR>.

* Once data set selection is made, another submenu, which requests data type selection, is displayed. Submenu options include:

          BAD_CODEBOOK: Selects data sets compressed using bad or
                        inappropriate codebooks.
           DESERT_AREA: Selects data sets covering desert areas.
          DIFF_PALETTE: Selects data sets using nonstandard color
                        palettes.
                   END: Terminates program execution.
       HIGH_DISTORTION: Selects data sets containing high distortion
                        values.
            METRO_AREA: Selects data sets covering metropolitan areas.
         MOUNTAIN_AREA: Selects data sets covering mountain areas.
     NOARL_COMPRESSION: Selects data sets processed using NRL's
                        (formerly the Naval Oceanographic and
                        Atmospheric Research Laboratory, or NOARL)
                        compression method.
            PC_ANOMALY: Selects data sets containing paper chart
                        anomalies (e.g., folds, creases).
         PREVIOUS_MENU: Returns control to the previous menu.
              TEST_SET: Selects test data sets.
            WATER_AREA: Selects data sets covering water areas.

* Once the data type selection is made, another submenu which requests scale and zone selection, is displayed. Options for

B-4

every map scale and zone are included (e.g., TLM NP, TPC NT, JNC EQ).

Select the appropriate scale and zone for the desired archive data set by using the arrow keys to highlight and pressing <CR>.

The selected archive data will now be displayed. Type <^Y> or <^C> when finished viewing (this will return you to the last sub-menu).

Another archive data set may be selected for viewing, or you may select one of the following options:
       PREVIOUS_MENU to return to the previous menu
                 END to end the session.

2 ADD_Datasets
This main menu option adds new data into the archive data set. The user must be logged in with the CAC account in order to use this option. Only compressed data can be added to the archive data set. The addition is performed by copying compressed data from a current processing thread to the archive data set area.

* A submenu is displayed for data set selection. Submenu options include CAC, CNC, or DLMS data sets (at present, only CAC is implemented).

    To select a CAC data set use the arrow keys to highlight the CAC option and press <CR>.

* Once data set selection is made, another submenu which requests data type selection is displayed.   Submenu options include:

         BAD_CODEBOOK: Selects data sets compressed using bad or
                       inappropriate codebooks.
          DESERT_AREA: Selects data sets covering desert areas.
         DIFF_PALETTE: Selects data sets using nonstandard color
                       palettes.
                  END: Terminates program execution.
      HIGH_DISTORTION: Selects data sets containing high distortion
                       values.
           METRO_AREA: Selects data sets covering metropolitan areas.
        MOUNTAIN_AREA: Selects data sets covering mountain areas.
    NOARL_COMPRESSION: Selects data sets processed using NRL's
                       (formerly the Naval Oceanographic and
                       Atmospheric Research Laboratory, or NOARL)
                       compression method.
           PC_ANOMALY: Selects data sets containing paper chart
                       anomalies (e.g., folds, creases).
        PREVIOUS_MENU: Returns control to the previous menu.
             TEST_SET: Selects test data sets.
           WATER_AREA: Selects data sets covering water areas.

* Once the data type selection is made, another submenu which requests scale and zone selection, is displayed. Options for every map scale and zone are included (e.g., TLM NP, TPC NT, JNC EQ).

    Select the appropriate scale and zone for the desired archive data set by using the arrow keys to highlight and pressing <CR>.

* Specify the input processing thread (e.g., A5B), color palette number (e.g., 133), and lower left-hand row/column numbers of the data to be added.

    Addition takes about 1 minute, then the prompt returns to the last submenu.

2 DELETE_Datasets

This menu option is used to delete data from the archive data set. The user must be logged in with the CAC account in order to use this option.

* A submenu is displayed for data set selection. Submenu options include CAC, CNC, or DLMS data sets (at present, only CAC is implemented).

    To select a CAC data set use the arrow keys to highlight the CAC option and press <CR>.

* Once data set selection is made, another submenu, which requests data type selection, is displayed. Sub_menu options include:

BAD_CODEBOOK:	Selects data sets compressed using bad or inappropriate codebooks.
DESERT_AREA:	Selects data sets covering desert areas.
DIFF_PALETTE:	Selects data sets using nonstandard color palettes.
END:	Terminates program execution.
HIGH_DISTORTION:	Selects data sets containing high distortion values.
METRO_AREA:	Selects data sets covering metropolitan areas.
MOUNTAIN_AREA:	Selects data sets covering mountain areas.
NOARL_COMPRESSION:	Selects data sets processed using NRL's (formerly the Naval Oceanographic and Atmospheric Research Laboratory, or NOARL) compression method.
PC_ANOMALY:	Selects data sets containing paper chart anomalies (e.g., folds, creases).
PREVIOUS_MENU:	Returns control to the previous menu.
TEST_SET:	Selects test data sets.
WATER_AREA:	Selects data sets covering water areas.

* Once the data type selection is made, another submenu which requests scale and zone selection, is displayed. Options for every map scale and zone are included (e.g., TLM NP, TPC NT, JNC EQ).

  Select the appropriate scale and zone for the desired archive data set by using the arrow keys to highlight and pressing <CR>.

* A prompt will ask if you are sure you want to delete a dataset. The data will be deleted when "Yes" is entered, otherwise the data will NOT be deleted.

  Control returns to the last submenu.

2 END
  Terminates program execution.

UNCLASSIFIED

APPENDIX C
TOPIC FILE BITMAPS.HLP

The following text and subtopics appear when **Bitmaps** is selected as
an **MDFFHELP** topic:

Bitmaps

Bitmaps are used by certain programs within the MDFF software
suite. All bitmap files are stored in the directory
MDFF_SYSTEM:[BITMAPS] and have the same format.

The Bitmap format uses the first 16 bytes (first 4 longwords)
to contain the minrow, mincol, maxrow, and maxcol
of the data that are being described. All remaining bytes
contain the actual 1 and 0 values.

Additional information available:

File_Names Summary_Files

The following text comprises this **MDFFHELP** topic file. Note that
subtopics begin with the key "2", which is located in column 1.

1 Bitmaps
 Bitmaps are used by certain programs within the MDFF software
 suite. All bitmap files are stored in the directory
 MDFF_SYSTEM:[BITMAPS] and have the same format.

 Bitmap format uses the first 16 bytes (first 4 longwords)
 to contain the minrow, mincol, maxrow, and maxcol
 of the data that are being described. All remaining bytes
 contain the actual 1 and 0 values.

2 File_Names
 The file naming convention for the different types of bitmaps are
 as follows:

    Type: CAC Bitmaps

    Name: a_b_c.TEMPLATE;d

    where
            a = CD_ID   Extracted from the CAC CD-ROM

            b = SCALE   Chart scale

```
        c = ZONE            0-4  possible TS zones

        d = VERSION         VAX/VMS file version number

Example:  CD-1991-A-MAP2-10007_2_3.TEMPLATE;1


   Type:  Generic Bitmaps

   Name:  a_b_c.TEMPLATE;d

   where
        a = Bitmap name     Determined by the user

        b = SCALE           Chart scale

        c = ZONE            0-4  possible TS zones

        d = VERSION         VAX/VMS file version number

Example:  MY_BITMAP_2_2.TEMPLATE;1


   Type:  AOD Template Bitmaps

   Name:  a_b_c_d.TEMPLATE_e;f

   where
        a = AOD name        Determined by the user

        b = SIDE            A, B, or C possible sides
                            (Note:  Side C denotes both side A and B)

        c = SCALE           Chart scale

        d = ZONE            0-4  possible TS zones

        e = Subdir Number   1-3 possible subdirectories

        f = VERSION         VAX/VMS file version number

Example:  ALASKAN_A_3_4.TEMPLATE_1;1
```

2 Summary_Files
Also contained in the MDFF_SYSTEM:[BITMAPS] directory are the AOD template summary files. These files contain, in ASCII format, information about a given AOD's template.

The file naming convention for these summary files are as follows:

   Type: AOD template summary files (ASCII File)

Name: a.LIS;b
where

      a = AOD name   Determined by the user (usually the same name as the AOD being summarized)

      b = VERSION   VAX/VMS file version number

EXAMPLE: ALASKAN.LIS;1

**APPENDIX D
TOPIC FILE CAC_PROCESSING.HLP**

TOPIC FILE: CAC_PROCESSING.HLP

The following text and subtopics appear when **CAC_Processing** is
selected as an **MDFFHELP** topic:

```
CAC_Processing

This topic covers the processing of CAC data.  Both the CAC
and CNC have the same source data format, ADRG. Thus, their
processing steps are very similar.

See the CNC_Specifics topic for CNC specific information.

Additional information available:

Overview    INITIALIZATION          PASS1       PASS2       PASS3A
PASS3B      WRAPUP      BUILD_ISO_IMAGE         Display_Programs
Segment_Files           Status_Files
```

The following text comprises this **MDFFHELP** topic file. Note that
subtopics begin with the key "2", which is located in column 1.

1 CAC_Processing

   This topic covers the processing of CAC data. Both the CAC and
   the CNC have the same source data format, ADRG. Thus, their
   processing steps are very similar.

   See the CNC_Specifics topic for CNC specific information.

2 Overview

   Interfacing with CAC processing is conducted through the
   user interface CHART_CONTROL. CHART_CONTROL sets control
   information in various status files (see Status_Files) that is
   in turn interpreted by the PROCESS_CHART program.

   A general overview of the processing steps is shown below:

                                    COMPRESSED CHART
                                    CORE
                                   /
INITIALIZATION---ADRG--[PASS1]-- \   FILLED SEGMENTS -----
                                   \   /
                                    EDGE --[PASS2]--
                                                 \
                                    UNFILLED SEGMENTS -----

                              D-3

```
              COMPRESSED
                CHART
---[PASS3A]---|--
              \
               \                  COMPRESSED
                \                    CHART
-------------------[PASS3B]-----|--------[WRAPUP]----[ODI IMAGE
                                                      GENERATION]
```
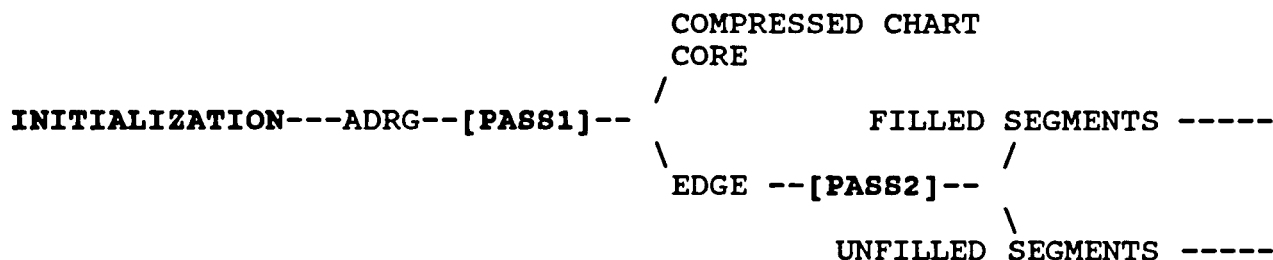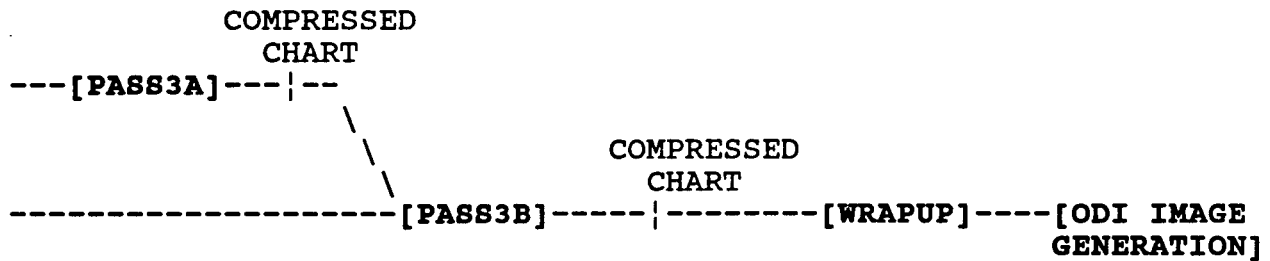
2 INITIALIZATION

    Beginning with the CHART_CONTROL INIT_BUILD option, a process
thread is created, logical names are built, and the generic
processing queue is established.

2 PASS1

    Beginning with the CHART_ODI START_PASS1 option, PASS1
attempts to build codebook procedures and to compress all
CORE segments.

    CHART_ODI options SELECT_READERS and VALIDATE_READERS are
used to begin processing ADRG data from CD-ROM for inclusion
into a CAC.

2 PASS2

    Beginning with the CHART_ODI START_PASS2 option, PASS2
attempts to fill remaining UNFILLED EDGE segments.
Due to lack of data, it may not be possible to fill all
EDGE segments. UNFILLED segments are compressed in PASS3B.

    CHART_ODI options SELECT_READERS and VALIDATE_READERS are
used for reading ADRG data from CD-ROM.

2 PASS3A

    Beginning with the CHART_ODI START_PASS3 option, PASS3
builds codebooks for and compresses FILLED segments from
PASS2 (or PASS1 when PASS2 in not performed).

2 PASS3B

    Compresses failed codebook segments from PASS3A and any
remaining UNFILLED segments (e.g. outer EDGE segments).
There are CHART_ODI options for chart build processing
to pause, stop, resume, and recover.

2 WRAPUP

Determines palette boundaries, and creates CD Coverage, Palette Coverage, Area Source, and Area DR files. Deletes extraneous files.

CHART_CONTROL option START_CHART_WRAPUP performs these actions.

2 BUILD_ISO_IMAGE

Creates an ISO 9660 Image from applicable files. CHART_CONTROL option START_ISO_IMAGE_BUILD performs this action.

After generation, the ISO 9660 image is copied to magnetic tape and sent to the mastering facility (to be produced as a CD-ROM). The command file that performs tape initialization and file copy is located at and named
MDFFEXE:CHART_ISO_IMAGE_TO_TAPE.COM

2 Display_Programs

Programs are available for displaying data on workstations (VAX/VWS, VAX/GPX, and IVAS) and hardcopy devices. These programs are described in the CAC_PROGRAM_DESCRIPTIONS topic.

2 Segment_Files

Naming conventions for downsampled and compressed files have been established. Each convention is described as a subtopic.

3 Downsampled_Segments
Downsampled segment files use the following naming convention:

                directory:[Rsnnnnn]12345678.90z

The "directory" will usually be "CHART_SEGMENTS".

For codebook repair, the string "directory:["
is replaced by the string "MDFF_SCRATCH:[CB_REPAIR."

Note the placement of the periods and brackets, they are significant!

The "s" is the sign of the row number. "0" for positive,
                                          "1" for negative.

The "nnnnn" is the row number to five whole digits.

The "12345678" is the first eight digits of the key name,

which is computed from the row and column numbers.

The "90" is the last two digits of the key name computed from the row and column numbers.

The "z" is the TS zone number (0 through 4).

4 Examples
For a downsampled segment on row 130 and column -378 in the equatorial zone (TS zone 2):

The segment is stored in the current processing thread's area for downsampled data with directory and file names:

        CHART_SEGMENTS:[R000130]01552137.532

The segment is stored in the current processing thread's area for codebook repair with directory and file names:

        MDFF_SCRATCH:[CB_REPAIR.R000130]01552137.532

3 Compressed_Segments
Compressed segment files use the following naming convention:

        CHART_ODI_DISK:[MAPx.PAkkkkmm.Rsnnnnn]12345678.90z

For codebook repair, the string "CHART_ODI_DISK:[MAPx.PAkkkkmm." is replaced by the string   "MDFF_SCRATCH:[CB_REPAIR."

Note the placement of the periods and brackets, they are significant!

The "x" is the TS scale (0 through 5).

The "kkkk" is the palette (i.e., PA) number of the palette used to compress this data.

The "mm" is an arbitrary sequence number assigned to each palette subdirectory. The order is dependent on the order that the palette subdirectories are created (ie. first: 01, second: 02, third: 03, and so on).

The "s", "nnnnn", "12345678", "90" and the "z" are the same as the downsampled filenames.

4 Examples
For a compressed segment on row 130 and column -378 in the equatorial zone (TS zone 2):

D-6

The data is stored in the current processing thread's area for compressed data with directory and file names:

        CHART_ODI_DISK:[MAP3.PA013001.R000130]01552137.532

The data is stored in the current processing thread's area for codebook repair with directory and file names:

        MDFF_SCRATCH:[CB_REPAIR.R000130]01552137.532

2 Status_Files

Status files are used to provide current information about CD-ROM readers and CAC processing.

The availability of CD-ROM readers is provided through the file CDREADER_STATUS.DAT.

The various stages of CAC processing are controlled by the status files: CHART_STATUS.DAT and CODEBOOK_STATUS.DAT.

The CHART_STATUS.DAT file is the master control file. It contains a header record and one status record for each ADRG CD-ROM input for processing.

A CODEBOOK_STATUS.DAT file is used to log information pertaining to the codebook build/compression phase of PASS1 and PASS3A.   There is one of these for each ADRG CD-ROM and one for each PA processed in PASS3A.

3 CDREADER_STATUS

This file resides in the MDFF_SYSTEM:[DATA] directory and is accessed through the SHOW_CDREADER_STATUS utility. Updates are

made when one of the following statements is true:

    A CD reader is mounted using the MOUNT_CDROM utility
    A CD reader is dismounted using the DISMOUNT_CDROM utility

For proper operation and maintenance, it is imperative that the MOUNT_CDROM and DISMOUNT_CDROM utilities be used.

3 CHART_STATUS

This file resides in the MDFF_SCRATCH:[000000] directory for the current processing thread. The file format is described in the file MDFFEXE:CHART_STATUS.INC.

4 Header_Record

These are the mnemonics, with their values, for fields in the header record which are used to control CAC processing.

Values for "PROCESSING_STATE":

NO_OP	0
INITIALIZING_BUILD	1
PASS1_IN_PROGRESS	2
PASS2_IN_PROGRESS	3
BUILDING_SEGMENT_LISTS	4
BUILDING_PASS3A_PROCS	5
BUILDING_PASS3A_CBS	6
BUILDING_PASS3B_PROC	7
COMPRESSING_PASS3B_SEGMENTS	8
BUILDING_ID_DIR_FILES	9
BUILDING_CHART_ISO_IMAGE	10

Values for "LAST_PHASE_COMPLETED":

NO_OP	0
BUILD_INITIALIZED	1
PASS1_COMPLETE	2
PASS2_COMPLETE	3
SEGMENT_LISTS_BUILT	4
PASS3A_PROCS_BUILT	5
PASS3A_CBS_COMPLETED	6
PASS3B_PROC_BUILT	7
PASS3B_SEGMENTS_COMPRESSED	8
ID_DIR_FILES_BUILT	9
CHART_ISO_IMAGE_BUILT	10
CHART_BUILD_COMPLETE	11

Values for "CONTROL":

NO_OP	0
CHART_BUILD_INIT	1
CHART_BUILD_PASS1	2
CHART_BUILD_PASS2	3
CHART_BUILD_PASS3	4
CHART_BUILD_PASS3_CONT	5
CHART_BUILD_WRAPUP	6
CHART_BUILD_ISO_IMAGE	7
CHART_BUILD_RESUME	8
CHART_BUILD_PAUSE	9

4 Status_Record

These are the mnemonics, with their values, for fields in the header record which are used to control CAC processing.

Values for "PROCESSING_STATE":

NO_OP	0
READING_ADRG	1
DOWNSAMPLING_ADRG	2
BUILDING_ADRG_CB_PROCS	3
SUBMITTING_ADRG_CB_PROCS	4
BUILDING_ADRG_CBS	5

Values for "LAST_COMPLETED_PHASE:

NO_OP	0
ADRG_READ	1
ADRG_DOWNSAMPLED	2
ADRG_CB_PROCS_BUILT	3
ADRG_CB_PROCS_SUBMITTED	4
ADRG_CBS_BUILT	5
CD_COMPLETE	6
TO_BE_REPROCESSED	7

Values for "LEGEND_STATUS":

NO_OP	0
LEGEND_START	1
LEGEND_PROCESSING	2
LEGEND_FINISHED	3
LEGEND_COMPLETE	4

3 CODEBOOK_STATUS

This file resides in the MDFF_SCRATCH:[CDmnnnnn] directory for the "nnnnn" ADRG CD-ROM being processed. The file format is described in the file MDFFEXE:CODEBOOK_STATUS.INC.

TOPIC FILE: CAC_PROGRAM_DESCRIPTIONS.HLP

The following text and subtopics appear when CAC_Program_Descriptions
is selected as an **MDFFHELP** topic:

CAC_Program_Descriptions

  This topic contains functional descriptions of executable
  images. All executable images are located in the directory
  named MDFFEXE.

  For more detailed information about these programs, see
  topic CAC_SOURCE_CODE.

  Additional information available:

  Overview   ALGEBRAIC_REMAP   ALG_PAL_BUILD_DRIVER
  BUILD_PASS1_PROCEDURES   BUILD_PASS3A_PROCEDURES
  BUILD_PASS3B_PROCEDURES   CB_REPAIR_IVAS
  CHART_CONTROL   CHECK_CDREADER_STATUS_DRIVER
  CLEAR_CDREADER_STATUS_DRIVER DECODEKEY DELETE_PA_DIR_DRIVER
  Display_Programs   DOWNSAMPLE_CD   DS_SEGS
  DUMP_CODEBOOK_STATUS DUMP_COLPAL_DB   DUMP_DS_SEGMENTS
  ENCODEKEY FIND_CD_STATUS   LIST_CHART_STATUS
  PROCESS_CHART   READ_ADRG_DATA   SHOW_CDREADER_STATUS
  TEK   TRIM_ODI_STRUCT

  The following text comprises this **MDFFHELP** topic file. Note that
  subtopics begin with the key "2", which is located in column 1.

1 CAC_Program_Descriptions
    This topic contains functional descriptions of executable
    images. All executable images are located in the directory
    named MDFFEXE.

    For more detailed information about these programs, see
    topic CAC_SOURCE_CODE.

2 Overview
    A variety of executable images are used in CAC data processing.
    Brief functional descriptions of these images follow.

                    ALGEBRAIC_REMAP: Uses a given CAC color palette and
                                     individual RGB shades to produce
                                     an algebraic color palette.

                 ALG_PAL_BUILD_DRIVER: Builds algebraic palettes for very

large scale data.

BUILD_PASS1_PROCEDURES: Builds command files for codebook
generation/data compression.

BUILD_PASS3A_PROCEDURES: Builds command files for FILLED
segment codebook/compressions.

BUILD_PASS3B_PROCEDURES: Builds command file for UNFILLED
segment compression and for failed
PASS3A codebook compressions.

CB_REPAIR_IVAS: Repairs segments using the
appropriate compression method
(Honeywell or NRL)
and the IVAS monitor for display.

CHART_CONTROL: User interface to CHART ODI
processing.

CHECK_CDREADER_STATUS_DRIVER: Checks the availability of a CD-ROM
reader. Is used by the MOUNT_CDROM
utility

CLEAR_CDREADER_STATUS_DRIVER: Clears the status of a CD-ROM reader
to make the reader available for
use. Is used by the DISMOUNT_CDROM
utility.

DECODEKEY: Decodes a segment key into a set of
row/column numbers.

DELETE_PA_DIR_DRIVER: Deletes an entire palette (PA)
directory, for specific TS zone.

Display_Programs: A suite of programs is available for
displaying data on workstations and
hardcopy devices.

DOWNSAMPLE_CD: Converts input ADRG data from the
ARC projection system to the TS
projection system.

DS_SEGS: Downsamples TS segments from ADRG
data that are stored on CD-ROM.

DUMP_CODEBOOK_STATUS: Lists information about the
codebooks for a specific CD-ROM or
palette.

DUMP_COLPAL_DB: Lists information about each color
palette.

E-4

DUMP_DS_SEGMENTS: Lists information about the downsampled segments of a CD.

ENCODEKEY: Encodes a set of row/column numbers into a key, which is used to name a segment file.

FIND_CD_STATUS: Checks all processing threads for the UNAVAILABILITY of CD-ROM readers.

LIST_CHART_STATUS: Lists contents of the current or specified processing thread's CHART_STATUS file.

PROCESS_CHART: Used as a continuing process that monitors and maintains CHART ODI processing.

READ_ADRG_DATA: Copies ADRG data from CD-ROM to magnetic disk.

SHOW_CDREADER_STATUS: Displays the status of CD-ROM readers, availability and owner process names (for unavailable readers). Used by the SHOW_CDREADER_STATUS utility.

TEK: Prints file with extensions TEK and IMAGE on the TEKTRONIX printer.

TRIM_ODI_STRUCT: Moves designated compressed and downsampled segments from a specified processing thread into a trim directory.

2 ALGEBRAIC_REMAP

Takes a CAC color palette (e.g., MDFF_SYSTEM:[COLPAL]PAOxxx.PALETTE) that has been supplied as input and applies individual RGB shade levels to remap the CAC palette's colors into the closest corresponding algebraic values. The RGB shade levels (e.g. red=8, green=6, blue=5 ==> 8x6x5 = 240 colors) are used to divide RGB space into small cubes. The centroid value of each cube is then compared with the input palette's values. The closest centroid value (in euclidean distance) replaces each input palette color.

This program is used for remapping CAC palettes into lower color levels. For example, entering RGB shades of 4,4,4 will remap a 240 entry color palette into 64 colors. The output palette can then be used to display CAC data at this lower color resolution.

ALGEBRAIC_REMAP is invoked using the following syntax:

$ RUN/NODEB V2_DIR:ALGEBRAIC_REMAP

3 Input

ALGEBRAIC_REMAP issues the following input prompts.

CAC palette disk name	Enter the symbol or name of the device containing the input CAC palette file. For example, MDFF_SYSTEM
CAC directory path	Enter the symbol or directory path for the directory containing the input CAC palette file. Brackets must not be included. For example, COLPAL
Palette filename	Enter the CAC palette file name. For example, PA0xxx.PALETTE where xxx is a palette ID number.

The following prompts are used to obtain RGB shade levels.
Note that RxGxB input values cannot exceed 240 (i.e., only 240 possible colors can be created).

Red shades	Enter an integer to represent a RED shade
Green shades	Enter an integer to represent a GREEN shade.
Blue shades	Enter an integer to represent a BLUE shade.

The user is also prompted for an output path name and names for the output (algebraic) colormap, palette, and color conversion files.

Output disk name	Enter the symbol or name of the device to contain the output files. For example, MDFF_SYSTEM
Output directory path	Enter the symbol or directory path for the directory to contain the output files. Brackets must not be included. For example, COLPAL
Algebraic palette filename	Enter the algebraic palette file name using the CAC palette format. For example, PA0xxx.PALETTE where xxx is a palette ID number.
Algebraic colormap filename	Enter the algebraic colormap file name

E-6

using the CAC colormap format and .DAT
extension. For example, PA0xxx.DAT
where xxx is a palette ID number.

Color conversion filename Enter the conversion file name using
using the CAC colormap format and .CNV
extension. For example, PA0xxx.CNV
where xxx is a palette ID number.
This file is used to remap 24-bit
color values to 8 bit.

3 Output

Program output consists of the following files, each of which is
written on the device and directory that were specified as
program input.

Algebraic palette file   The algebraic palette file name
uses the CAC palette format.
For example, PA0xxx.PALETTE
where xxx is a palette ID number.

Algebraic colormap file   The algebraic colormap file name
uses the CAC colormap format and .DAT
extension. For example, PA0xxx.DAT
where xxx is a palette ID number.

Color conversion file   The conversion file name using
the CAC colormap format and .CNV
extension. For example, PA0xxx.CNV
where xxx is a palette ID number.
This file was used to remap 24-bit
color values to 8 bit.

2 ALG_PAL_BUILD_DRIVER

This program is used to build algebraic palettes for very large
scale data sets where standard color palette builds are
impractical. Familiarity with color mechanics and RGB space is
assumed.

The algebraic palette that is built is based on RGB shade levels
that are supplied as program input. For example, using inputs
where Red=8, Green=6, and Blue=5, will produce (8 x 6 x 5), or 240
distinct colors. The shade levels are used to divide RGB space
into small "boxes," where the centroid value of each box is used
as a color palette entry.

The length of each box on an axis of RGB space is computed using
each shade level where:

box length = (total number of colors) / shade level

E-7

Hence, box length in RGB space for the red axis = 256/8; for the green axis = 256/6; and for the blue axis = 256/5.

The centroid value is simply half the value of the length of the box in RGB space. Hence,

```
the red axis   = (256/8) / 2 = 16
the green axis = (256/6) / 2 = 21
the blue axis  = (256/5) / 2 = 26
```

3 Input

The following prompts are used to obtain RGB shade levels. Note that (Red x Green x Blue) input shade levels cannot exceed 240 (i.e., only 240 colors can be created).

Red shade	Enter an integer to represent a RED shade level.
Green shade	Enter an integer to represent a GREEN shade level.
Blue shade	Enter an integer to represent a BLUE shade level.

The user is also prompted for an output device name, path name and names for the algebraic palette, colormap, and color conversion files.

Output disk name	Enter the symbol or name of the device to contain the output files. For example, MDFF_SYSTEM
Output directory path	Enter the symbol or directory path for the directory to contain the output files. For example, COLPAL
Algebraic palette filename	Enter the algebraic palette file name using the CAC palette format. For example, PA0xxx.PALETTE where xxx is a palette ID number.
Algebraic colormap filename	Enter the algebraic colormap file name using the CAC colormap format and .DAT extension. For example, PA0xxx.DAT where xxx is a palette ID number.
Color conversion filename	Enter the conversion file name using using the CAC colormap format and .CNV extension. For example, PA0xxx.CNV where xxx is a palette ID number. This file is used to remap 24-bit color values to 8 bit.

3 Output

Program output consists of the following files: the algebraic palette, algebraic colormap, and a color conversion file. Each of these files is described below.

Algebraic palette file	The algebraic palette file name uses the CAC palette format. For example, PA0xxx.PALETTE where xxx is a palette ID number.
Algebraic colormap file	The algebraic colormap file name using the CAC colormap format and .DAT extension. For example, PA0xxx.DAT where xxx is a palette ID number.
Color conversion file	The conversion file name using using the CAC colormap format and .CNV extension. For example, PA0xxx.CNV where xxx is a palette ID number. This file is used to remap 24-bit color values to 8 bit.

These output files are stored in binary format and may be read using the "C" functions that are located in CAC_UTILS.C: READ_PALETTE.C, READ_COLORMAP.C, and READ_COLOR_CONVERSION.C.

2 BUILD_PASS1_PROCEDURES

Constructs PASS1 codebook build/compression procedures for CORE segments and creates/modifies the related Codebook_Status file. The Codebook_Status File contains an entry for every build request that was generated. Codebooks are generated using 9 of 25 CORE segments.

2 BUILD_PASS3A_PROCEDURES

Constructs PASS3A codebook build/compression procedures for FILLED (EDGE) segments and creates/modifies the related Codebook_Status File. The Codebook_Status File contains an entry for every build request that was generated.

Builds PA codebooks using nine (of nine) segments.

2 BUILD_PASS3B_PROCEDURES

Constructs all codebook build/compression procedures for UNFILLED (EDGE) segments and for failed PASS3A codebook compressions.

Uses the nearest segment's codebook method for compression.

2 CB_REPAIR_IVAS

Determines the compression type of the data to be repaired and,

using the appropriate compression method, repairs it. The two compression methods are Honeywell and NRL.

Bad segments are identified during the review process and are downsampled into the CB_REPAIR area for repair. CB_REPAIR_IVAS displays the bad segment on the IVAS monitor and uses a codebook, from a preselected segment, to again (via appropriate method) compress the downsampled data. The result is displayed, and if acceptable, written to the CHART_ODI_DISK area replacing the bad segment.

3 Input_Output
  The following inputs are required during program execution:

        Enter the chart scale (0,1,2,3,4,5)
        Enter the zone (0,1,2,3,4)
        Enter the palette number
        Enter CD number
        Enter the row and column number (of the segment) to repair
        Enter the row and column (of a segment) to use for a
        codebook
        Does image pass (Y/N)?

  Outputs: If the newly compressed segment image is acceptable, it
           is written to the CHART_ODI_DISK area and is used to
           replace the previous bad segment.

3 Required_Subroutines
  The following routines are required for program execution.

           CB_REPAIR_IVAS: Driver program for repair subroutines.
            H_CB_REPAIR: Honeywell compression repair module.
            N_CB_REPAIR: NRL compression repair module.
                 READER: Module to read a downsampled data file.
      COMPRESS_SEG_NOARL: Module to compress a segment via NRL
                           compression.
     GET_CAC_DATA_24_BIT: Module to read 1 CAC segment and
                           decompress it to 24 bits.
               GET_INFO: Module in NRL compression to prompt user
                           for input.
          MAKE_DS_NAMES: Module to make the downsampled file names
                           for the READER module.
           GET_DS_DATA: Module to get 24-bit downsampled data.
              PUT_IMAGE: Module to put a 256 x 256 image onto the
                           IVAS screen.
       MAKE_ODI_SEGNAME: Module to make the CHART_ODI_DISK segment
                           name.
          GET_CODEBOOK: Module to get and decompress a segment
                           codebook.
      DISPLAY_CODEBOOK: Module to display a codebook on the IVAS
                           monitor.
            UNPACK_CAC: Module to unpack or decompress a CAC

E-10

```
                          segment.
            PRINTIO:  Module to print a image on the TEKTRONIX
                      printer.
      NOARL_TEKPRINT:  Translation driver for the TEKTRONIX
                      printer.
```

2 CHART_CONTROL

The CHART_CONTROL program is the main user interface for Version 2 CAC processing. This is a menu driven program. To select a function, use the arrow keys to highlight the desired function, then hit <CR>. To exit a menu (function) with no action taken, type <CTRL_Z>.

FUNCTION	ACTION
START_PASS1	Enables the processing of ADRG CD-ROMs for PASS1.
START_PASS2	Enables the processing of ADRG CD-ROMs for PASS2.
START_PASS3	Begins PASS3 processing phase. Builds the following files: UNFILLED_SEGMENTS.DAT, FILLED_SEGMENTS.DAT, and SEGMENTS.DAT. A mail message is sent to notify operators that the program DISPLAY_SEGMENTS must be run in order to check the validity of these files.
CONT_PASS3	Continues PASS3 processing by building and submitting codebook build/compression procedures for FILLED segments (PASS3A). Finally, UNFILLED segments and those segments remaining after PASS3A, are compressed using nearest segment's codebooks (PASS3B).
CHART_WRAPUP	Builds [ID] directory files and adds ODI information (i.e., inventory of processed ADRG CDs used to build the CAC) to the file named MDFF.DBASE.
SELECT_READERS	Select the previously validated ADRG CD-ROM(s) for processing. This phase adds the selected CD-ROM(s) to the CHART_STATUS database.
VALIDATE_READERS	Validate the input ADRG CD-ROM(s). Ensure that the CD-ROM is the right scale and area for inclusion into the current build. This phase only displays the CD-ROM contents. No

E-11

action is taken on the validated CD-ROM, the option SELECT_READERS adds this data.

RECOVER_BUILD   Recovers interrupted processing. For phases PASS1 and PASS3, CHART_STATUS entries in downsampling phase can be started either "cold" (i.e., from the very beginning) or "warm" (i.e., from where processing was prior to the interruption). CHART_STATUS entries previously submitted for building codebooks are resubmitted to processing queues as are entries that were not successfully completed prior to the interrupt.

MISCELLANEOUS   Miscellaneous functions to modify the CHART_STATUS database for the current build.

PAUSE_BUILD   Pauses CHART processing for the currently selected processing thread.

RESUME_BUILD   Resumes CHART processing for the currently selected processing thread.

INIT_BUILD   Initialize the CHART processing system for an ODI build.

STOP_BUILD   Stops CHART processing for the currently selected processing thread.

2 CHECK_CDREADER_STATUS_DRIVER

This program checks and updates the availability of a CD-ROM reader. It is executed through the MOUNT_CDROM DCL symbol.

MOUNT_CDROM executes a command file, named MOUNT_CDROM.COM, which invokes CHECK_CDREADER_STATUS_DRIVER.

The CDREADER_STATUS file contains an availability status for each CD-ROM reader. The MOUNT_CDROM utility checks CD-ROM reader availability by reading the CDREADER_STATUS file. If a specified CD-ROM reader is available for use, the CD-ROM reader is allocated and mounted and the status in CHART_STATUS is updated to show the reader as being unavailable for use. Therefore, it is imperative that the MOUNT_CDROM utility be used for allocating and mounting CD-ROM readers.

Note: CD-ROM readers may be mounted as being a FOREIGN device or an ISO image.

2 CLEAR_CDREADER_STATUS_DRIVER

This program clears the status of a CD-ROM reader (i.e., showing the reader as being available for use) and is executed

through the DISMOUNT_CDROM DCL symbol.

DISMOUNT_CDROM executes a command file, DISMOUNT_CDROM.COM which invokes CLEAR_CDREADER_STATUS_DRIVER.

The CDREADER_STATUS file contains an availability status for each CD-ROM reader. The DISMOUNT_CDROM utility updates the

CDREADER_STATUS file to reflect changes in CD-ROM reader availability. Therefore, it is imperative that DISMOUNT_CDROM utility be used for dismounting and deallocating CD-ROM readers.

2 DECODEKEY
This program prompts for a segment's key. This key is decoded to row and column numbers that are displayed as output.
Row and Column numbers are computed as follows:

row = mod [(key - 162018001),18001]
col = [(key - 162018001) - row / 18001]

2 DELETE_PA_DIR_DRIVER
Deletes an entire PA Directory, determined by the zone, for a specific processing thread. Perform the following steps to execute, the program:

* Set the processing thread for the data to be deleted.
* Type the command RUN/NODEB MDFFEXE:DELETE_PA_DIR_DRIVER

A message is displayed that advises the user to ascertain that the current processing thread is the proper processing thread for data deletion.

NOTE: ** This is very important - If the wrong processing thread
            is set, the wrong data could be deleted. **

The user is then prompted for the zone of the data to be deleted. All compressed segment data, contained within that zone, are deleted. In addition, the PA directory and all underlying subdirectories are deleted.

2 Display_Programs
This suite of programs displays data on a variety of devices.

3 Overview
This suite of programs displays data on a variety of devices. Hence, device names are usually included as part of the program name. Programs that display data on the International Imaging Systems image processor, IVAS, include the name IVAS. Programs

E-13

that display data on a DEC VAXstation monitor, running VAX
Workstation Software (VWS), include the name VWS.

DISPLAY_ADRG_IVAS: Displays ADRG data on the IVAS. Data are
usually read from CD-ROM.

DISPLAY_CAC_IVAS: Displays compressed segments on the IVAS.
Data are read from the CHART_ODI_DISK:

DISPLAY_CAC_PORTABLE: X-windows/MOTIF version of DISPLAY_CAC
that is portable to VAX/VMS systems.

DISPLAY_CAC_VWS: Displays compressed segments on a DEC
VAXstation running VWS.

MAPSTATION_DRIVER: Displays the area of coverage, for a
mapstation subdirectory, on a DEC
VAXstation running VWS.
A CalComp plot file, showing the area of
coverage, may also be created.

PLOT_CDS_VWS: Displays downsampled segments from ADRG CDs
with each segment represented by a
rectangle.

PLOT_ODI: Plots the area of a chart ODI build.
Enables the addition and/or deletion of
selected charts to the ODI build.

3 DISPLAY_ADRG_IVAS
This program is used to view ADRG data from CD-ROM on the IVAS
color monitor. The ADRG CD-ROM can be treated as a hard disk
with the exception that files cannot be deleted or created (as it
is mounted READONLY). Because the CD-ROM is accessed as a hard
disk drive, subdirectories and their files are accessible with
conventional VAX/VMS commands (e.g., SET DEFAULT).

DISPLAY_ADRG_IVAS requires the names of the Distribution
Rectangle (DR) and the General_Information_File (GEN) file.
Usually, the DR and the GEN file have the same name, with the GEN
file using a ".GEN" extension. The name of the DR is found by
obtaining a directory listing of the ADRG CD-ROMs' root
directory. The GEN file name can be found by obtaining a listing
of the DR directory. The example of program execution (see
subtopic EXAMPLE) shows how these file names are obtained. ADRG
CD-ROM TPC G-19D is used in the example. The DR name is TPUS0101
and the GEN file, found inside this directory, is named
TPUS0101.GEN. Once this information is known, DISPLAY_ADRG_IVAS
can be executed.

4 Input_Output
Perform the following steps to execute DISPLAY_ADRG_IVAS:

A. Place the CD-ROM in one of the CDROM readers and mount it using the MOUNT_CDROM command.

B. Enter the ADRG CD-ROM reader containing the ADRG CD-ROM.

C. Enter the DR name.

D. Enter the GEN file name. The extension ".GEN" may be omitted as this is the default extension).

E. ZDR selection: The number of ZDRs contained within the DR is displayed. Minimum and maximum values for latitudes and longitudes are displayed for the first ZDR. The user has the option to display the data in the current (in this case, the first) ZDR or skip to the next ZDR.

Selection of a ZDR is done by entering a lower or upper case 'Y' or by entering <RETURN>. To skip a ZDR, enter a lower or upper case 'N'.

Enter a starting coordinate and an ending coordinate (ADRG polar data is the exception where only one coordinate is entered and one image per input coordinate is displayed).

These points refer to the upper left and lower right corners of an area to display. Therefore, the upper left latitude must be greater than or equal to the lower right latitude and the upper left longitude must be less than or equal to the lower right longitude (if the two points are the same, only one screen of data will be displayed). Display proceeds from the upper left to the upper right until the right longitude is reached whereby the next frame returns to the left longitude and moves down in latitude.

Once the entire defined area has been displayed, step E is repeated for the next ZDR. DISPLAY_ADRG_IVAS terminates when there are no more ZDRs to display.

F. Placement of coordinates on the IVAS monitor. There are nine choices:
  1) Top left corner (TL)
  2) Top center (TC)
  3) Top right corner (TR)
  4) Center left (LC)
  5) Center of screen (CC)
  6) Center right (RC)
  7) Bottom left corner (BL)
  8) Bottom center (BC)
  9) Bottom right corner (BR)

Placement is selected by entering (lower case or upper case) any of the above options in parenthesis, TL through BR. Note

E-15

that the prompt ends with (TL/BR)[CC]. Where

>   (TL/BR) indicates valid selections are between the first
>   and last options (inclusive). Any other values are
>   invalid.

>   [CC] indicates that option 5 (Center of screen) is the
>   default option.

After an image is displayed on the IVAS monitor the user must
enter one of the following options:

>   Enter <RETURN> to continue to the next image
or
>   Enter <^Z> to generate an image on disk that will then
>   be printed on the TEKTRONIX printer.

If the print option is selected, the user is prompted for an
output file name. When generating images on disk, use the
IIS_IMAGES:[username] directory and delete all files that are
no longer needed, as the files tend to accumulate and take up
valuable disk space. Once the file has been submitted to the
printer, display resumes from where it left off.

G. Early termination. The user may terminate execution before the
   endpoint is reached with the following commands:

>   Enter <^Z>   When prompted to "continue or create/print an
>   image".

>   Enter <^Z>   When prompted for an image file name.

4 Example
  The following is an example of program execution

```
$ @mdffexe:mount_cdrom cdrom01
Mounting CDROM01
%I-ALLOC, _OSPREY$DUB0: allocated
%I-MOUNTED, ARC1 TPCXXG1 mounted on _OSPREY$DUB0:
$ dir cdrom01:[000000]
Directory OSPREY$DUB0:[000000]

TESTPA01.CPH;1        TPUS0101.DIR;1        TRANSH01.THF;1

Total of 3 files.

$ dir cdrom01:[tpus0101].gen
Directory OSPREY$DUB0:[TPUS0101]

TPUS0101.GEN;1

Total of 1 file.
```

```
$ run/nodebug v2_dir:display_adrg_ivas

ENTER Disk name: cdrom01
ENTER Directory path name: tpus0101
ENTER GEN file name: tpus0101
There are  2 ARC zones on this cd

northlat:  32.46063613891602 degrees = 32 deg 27 min 38.29 sec
southlat:  31.99982833862305 degrees = 31 deg 59 min 59.38 sec

westlong:-116.00003814697266 degrees = -116 deg  0 min  0.14 sec
eastlong:-108.99995422363281 degrees = -108 deg 59 min 59.84 sec

Use data from this area (Y/N)[Y]? n
northlat:  35.66695785522461 degrees = 35 deg 40 min  1.05 sec
southlat:  31.99982833862305 degrees = 31 deg 59 min 59.38 sec

westlong:-116.00016021728516 degrees = -116 deg  0 min  0.58 sec
eastlong:-108.99983978271484 degrees = -108 deg 59 min 59.42 sec

Use data from this area (Y/N)[Y]?
Starting position (Upper right  corner)

Enter coordinates as either decimal or deg,min,sec
Examples: 90.000 or 90 00 00 (use - for south/west
          latitude/longitude

Enter latitude to view: 35
Enter longitude to view: -115
Ending position (Lower left corner)

Enter coordinates as either decimal or deg,min,sec
Examples: 90.000 or 90 00 00 (use - for south/west
          latitude/longitude

Enter latitude to view: -34.5
Enter longitude to view: -114.5

Input coordinates are for:
Top Left corner   (TL)
Top Center        (TC)
Top Right corner  (TR)
Left Center       (LC)
Center            (CC)
Right Center      (RC)
Bot Left corner   (BL)
Bot Center        (BC)
Bot Right corner  (BR)  (TL/BR)[CC]:

Latitude, Longitude    34.9872 -115.0508
Hit <CRT> to continue or <CTRL_Z> to create/print image: <CRT>
```

```
Latitude, Longitude    34.9872 -114.5178
Hit <CRT> to continue or <CTRL_Z> to create/print image:
<CTRL_Z>

Enter output file name
(<CTRL_Z> not to print): iis_images:[riedlinger]test.image

Writing image to disk!!
Would you like full prompting? (Y or N) n

Converting image to TEKTRONIX format

Printing file!!!


Latitude, Longitude    34.9872 -113.9848
Hit <CRT> to continue or <CTRL_Z> to create/print image: <CTRL_Z>


Enter output file name
(<CTRL_Z> not to print): <CTRL_Z>

End DISPLAY_ADRG_IVAS

$ @mdffexe:dismount_cdrom cdrom01
Dismounting CDROM01
```

3 DISPLAY_CAC_IVAS
   Reviews compressed data on the IVAS monitor. Data are read from
   the CHART_ODI_DISK area and displayed on the IVAS screen 16
   segments at a time.

4 Input_Output
 Inputs required for program execution include:

```
        Enter the map scale (0:8)
        Enter the palette number
        Enter the TS zone number
**      Segment Image (Y/N) [N]
                If Y will prompt: Color for lines (B/W) [B]
        Enter min row
        Enter max row
        Enter min col
        Enter max col
```

   Note:
   ** Grid lines may be drawn to visually "segment" the display.
      The default will not draw these lines. When in the lines
      are to be drawn (i.e., "Y" was entered), a prompt for line
      color is presented. The default line color is Black, the
      alternative color is White.

E-18

Output: Compressed segment data are displayed on the IVAS monitor
16 segments at a time.

4 Required_Subroutines
The following routines are called by DISPLAY_CAC_IVAS.

DISPLAY_CAC_IVAS: Main program.
READER: Module for reading the stream CAC file into memory.
GET_CAC_DATA_8_BIT: Module for obtaining 8-bit CAC data.
GET_POS: Module for setting up a position on the IVAS color monitor.
PRINTIO: Module for printing an image from the IVAS.
NOARL_TEKPRINT: TEKTRONIX printer driver for printing an image.
ADDTO_SCREEN_BUFFER: Module for adding one 256 x 256 image to the large print buffer.

3 DISPLAY_CAC_PORTABLE
This X-windows/MOTIF version is designed to be portable to various
VAX/VMS platforms. The name for this portable version is simply
DISPLAY_CAC. Instructions for reading and displaying CAC
data are listed as subtopics.

4 Getting_Started
The following items are required to install and execute
DISPLAY_CAC:

a. A copy of DISPLAY_CAC.EXE. This may be stored on floppy
diskette, 9-track tape, or 6250 drive. (i.e., on one of the
systems's storage devices)

b. Volume and File Structure (VFS-ISO9660) software,
serial #YT-GS001-01, Version 2.2. or
F11CD-ISO9660 software, serial #A-MRAAA-H8, July 1992.
NOTE: F11CD software requires VMS Operating System,
Version 5.5.

c.   CD-ROM reader hardware/software.

d.   X-windows/MOTIF graphics display software.

The following procedure is used to install DISPLAY _CAC software:

a.   Create a new directory (e.g., [.DISPLAY_CAC] to store the
program.

b.   Copy the DISPLAY_CAC.EXE program (from the floppy,
or 9-track tape) into the [.DISPLAY_CAC] directory.

4 Selecting_CAC
 Select a CAC CD-ROM that contains the desired area of coverage.
 The available area of coverage is depicted on the CAC's CD-ROM
 case.

> 1) Place the CAC CD-ROM to be viewed in the CD-ROM reader.
>
> 2) Mount the CD reader.

4 Program_Execution
> 1) Set default to the directory containing DISPLAY_CAC.EXE.
>
> 2) A symbol must be defined to execute DISPLAY_CAC. To
>   define the symbol, type the following command at the
>   operating system prompt:
>
>   display_cac == "$ XXXdisplay_cac.exe"
>
>   where XXX is a path to the [.DISPLAY_CAC] directory.
>
> 3) Execute DISPLAY_CAC by typing the command:
>
>   display_cac cdrom03:
>
>   where cdrom03: is the device name (i.e., CD-ROM reader)
>   as a command line argument.

4 Viewing_CAC
> 1) Execute DISPLAY_CAC (see subtopic Program_Execution).
>
> 2) A map of the entire world is displayed on the
>   screen. Click once on a center point of CAC data
>   to be viewed.
>
>> a.   The map data area of coverage that is available
>>   can be obtained from the CAC's CD-ROM case.
>>
>> b.   If the message 'No coverage available...try again.'
>>   appears when you click, try clicking again on a
>>   different area.
>
> 3) When a center point has successfully been selected, a
>   blank white screen will appear, and the program will
>   begin reading the map data from the CAC CD-ROM that has
>   been loaded in the CD_ROM reader. NOTE:: This could take
>   a minute or two.
>
> 4) After all the data has been read, the buttons
>   UP, DOWN, LEFT, RIGHT, EXIT and DISPLAY DATA will appear.
>   Click on the DISPLAY DATA button to display the
>   CAC data on the screen.

4 Traversing_CAC
1) Once the first screen of CAC data has been displayed
(see subtopic Viewing_CAC), more data located above,
below, to the left and right can be viewed by clicking
on the UP, DOWN, LEFT or RIGHT buttons, respectively.

2) Loading new data from the CAC CD-ROM could take
a minute or so.

4 Program_Termination
1) To exit the program, just click on the EXIT button.

2) This will terminate the program and return control
to the operating system.

3 DISPLAY_CAC_VWS
Displays Compressed Aeronautical Chart (CAC) segments on
VAXstations running VWS software.

4 Invocation
DISPLAY_CAC_VWS may only be executed on VAXstations running VAX
VWS software. Hence, workstations SPARK1 and SPARK3 are available
for program execution.

Perform the following steps in invoke DISPLAY_CAC_VWS:

 * Set the processing thread
 * Type the command   RUN/NODEB DISPLAY_CAC_VWS

4 Input_Output
DISPLAY_CAC_VWS requires the following input:

Zone number: Enter the TS zone number that
contains the compressed
segments to be displayed.

Continuous Viewing? (Y/N) [Y]: Enter 'Y' if more than one
screen of data is to be
displayed else enter 'N'.
The default is 'Y'.

Segment image? (Y/N) [Y]: Enter 'Y' to draw grid lines,
which separates individual
segments, within the image.
The default is 'Y'.

Start ROW (CTRL_Z to restart): Enter starting segment row
number. Typing (^Z) restarts

the input prompts.

Start COL (CTRL_Z to change rows): Enter starting segment column number.

End ROW   (CTRL_Z to restart): Enter end segment row number. Typing (^Z) restarts the input prompts.

End COL (CTRL_Z to change rows): Enter end segment column number. Typing (^Z) restarts the input prompts.

Note: The chart scale and PA number are obtained from the CHART_STATUS file.

Output consists of a 3-row by 4-column segment color display.

4 Required Subroutines
DISPLAY_CAC_VWS utilizes the following subroutines:

READ_CHART_STATUS_HEADER: Reads needed information from the CHART_STATUS.DAT file.

READ_PALETTE: Reads the RGB data from the palette color table.

LOAD_COLORTABLE2: Loads the RGB values from the palette color table onto the 3200 graphics color lookup table.

FIND_PA_FOR_ZONE: For a given scale and zone, returns to the PA number.

GET_SEGMENT_NAME: Builds directory path names for compressed segment files.

CAC_UTIL: Source code for standard "C" routines used throughout the MDFF software suite.

SPDEC: "C" routine used to DECOMPRESS compressed segment files.

3 MAPSTATION_DRIVER
MAPSTATION_DRIVER displays the area of coverage from a mapstation subdirectory on a VAXstation 3200 monitor. A CalComp plot file, showing the areas of coverage, may also be created (and is used for generating hardcopies).

Before program execution, mapstation subdirectory files must be

E-22

transferred from a mapstation onto the MDFF cluster. At program
execution, a mapstation subdirectory is read and one of the
following tasks is selected and performed:

    1.   Create a CalComp plot file of
        * the individual segments and
        * the outline of a mapstation subdirectory
          (i.e., a pilot plot)

    2.   Display on the 3200 color monitor
        * the individual segments and
        * the outline of a mapstation subdirectory
          (i.e., a pilot plot)

MAPSTATION_DRIVER is menu driven. The following menu options are
available, each of which is described as a subtopic:

    Enter:
          1 for a pilot plot
          2 for a plot of individual segments
          3 for displaying individual segments on 3200
          4 for displaying pilot plot on 3200
          5 testing polar plots

4 Program_Notes
* (^Z) is used to exit each menu and to terminate program
  execution.

* The CalComp plot file is opened in the main program.   All plot
  images are written into one output file which saves (expensive)
  plotter paper. Hence, one output plot file may contain several
  images, for example, a pilot plot image (option 1) and a plot
  of individual segments (option 2)

* Proper program termination via (^Z) must occur in order to have
  an EOF marker placed at the end of the plot file.   Without an
  EOF marker, the CalComp plotter will remain in a halted state.
  Once in a halted state, the plot must be terminated by using the
  terminal (located next to the plotter) with either the CANCEL or
  FLUSH command.

* File extensions are not required as input. For example, when
  opening a plot file for JOG scale plots, the full file name
  is JOB.CALC   Enter the file name without the extension: JOG

4 Pilot_Plot
This options creates an outline plot, called a pilot plot, of the
area of coverage by the mapstation directories. This plot will be
used by the aircraft pilots for AOD coverage purposes.

For each side of the AOD, all mapstation subdirectory files having the same chart scale should be plotted together. This will provide the total area of coverage (contained on the AOD), for a given scale, on one piece of paper.

Before this option is executed, the user must know the chart scale, zone, coverage for the entire plot area, and names of the mapstation subdirectory files.

5 Input

The following procedure must be repeated to include the area of coverage for each zone:

> A prompt will ask for the number of subdirectories in a particular zone. Enter (ctrl-z) to stop entering subdirectory names and to exit this procedure.

> After entering the number of subdirectories for a zone, program control loops to read that number of subdirectory names.

> Once all of the subdirectory names are read, MAPSTATION_DRIVER will prompt for the zone number

The following prompts are provided for input:

Lower latitude and longitude: Enter the lower left coordinates of the rectangular area of coverage.

Upper latitude and longitude: Enter the upper right coordinates of the rectangular area of coverage.

Scale: Enter the chart scale.

Title: Enter the pilot plot title.

Side of AOD: Enter "A" or "B" to indicate which side of the AOD is being used to store the data.

(Area of coverage procedure)
How many files are to be combined
for this zone:(ctrl-z)to quit): Enter the number of mapstation subdirectories to be included as coverage for this zone, or enter (ctrl-z) to exit.

Mapstation directory name: This prompt repeats until all

E-24

subdirectory filenames have been
entered.

Zone: Enter the zone number.

5 Output
The file naming convention for plot files is as follows:

AREA.CALC

where

AREA = A file name that is descriptive of the area of coverage.
CALC = The file extension, denoting the file is a CalComp plot
file.

4 Plot_of_Individual_Segments
This option creates a plot of individual segments that are
contained within a single mapstation subdirectory. The plot will
be used by NWC to review the AOD for accuracy.

This option also creates a listing for each plot that contains the
maximum and minimum latitude and longitude coordinates for each
row of segments. Subtopic output contains additional information
on output files.

Before this option is executed, the user must know the chart
scale, zone, and the area of coverage for each subdirectory. The
user must also know the subdirectory number.

NWC reviews each subdirectory separately. By using this approach,
it is easier to determine the subdirectory bounds and accuracy.
Hence, only one subdirectory per plot is allowed.

5 Input
The following prompts are provided for input:

Lower latitude and longitude: Enter the lower left coordinates
of the rectangular area of
coverage.

Upper latitude and longitude: Enter the upper right coordinates
of the rectangular area of
coverage.

Scale: Enter the chart scale.

Zone: Enter the zone's area of coverage

Title: Enter the pilot plot title.

E-25

Side of AOD: Enter "A" or "B" to indicate which
side of the AOD is being used to
store the data.

Subdirectory number:

Plot row_col: (which really means: would you
like to label the row and col, and
create a listing of the min/max
latitude/longitude of each row).
Enter "Y" or "N"

5 Output
The file naming convention for plot files is as follows:

AREA.CALC

where

AREA = A file name that is descriptive of the area of coverage.
CALC = The file extension, denoting the file is a CalComp plot
file.

This option also creates a listing for each plot that contains the
maximum and minimum latitude and longitude coordinates for each
row of segments.   The file naming convention for the plot file
listings is as follows:

AREA.LIS

where

AREA = A file name that is descriptive of the area of coverage.
LIS = The file extension, denoting the file is a listing of
a plot file.

4 Displaying_Individual_Segments
This option allows the user to display a mapstation subdirectory
on the VAXstation 3200. This option is primarily used to obtain
the area of coverage and to view missing segments within a single
mapstation subdirectory.

5 Input
The following prompts are provided for input:

Scale: Enter the chart scale.

Zone: Enter the zone's area of coverage.

Mapstation filename: Enter the mapstation subdirectory name.

Once these prompts are answered, the last five row and column

numbers are displayed. At times, extraneous segments not actually part of the area of coverage are created. These segments are detected when their row/column numbers are out of sequence with valid segment row/column numbers.

* If the last set of row/column numbers are out of sequence, then they should be considered invalid segments and eliminated. Invalid segments are eliminated by simply reducing the number of segments to be viewed (hence, only including valid segments).

* The number of segments to be viewed should be changed to include the last valid set of row/column numbers.

4 Displaying_Pilot_Plot
This option allows the user to display the bounds of a mapstation subdirectory on the 3200. This option is primarily used to view the outline of a single mapstation subdirectory.

5 Input
The following prompts are provided for input:

Do you want to set the bounds
for this subdirectory: [n]   The default is "n" for NO.
            If "y" for YES:
                Lower latitude and longitude: Enter the lower left
                                              coordinates of the
                                              rectangular area of
                                              coverage.

                Upper latitude and longitude: Enter the upper right
                                              coordinates of the
                                              rectangular area of
                                              coverage.

                    Scale: Enter the chart scale.

                    Zone: Enter the zone's area of coverage.

    Enter the number of files
    to read for this zone: Enter the number of files, contained
                           within the zone, that are to be included
                           in the area of coverage.

        Mapstation filename: Enter the mapstation subdirectory name.

4 Polar_Plots
This option will create either a plot file of the individual segments or an outline of the bounds for a single subdirectory of polar data.

The individual segment plot will be used by NWC to review the AOD

E-27

for accuracy. The bounds plot will be used by aircraft pilots for
coverage purposes.

All latitudes and longitudes are hardcoded.

As of March 12, 1992, this program reads from a processing thread.
Once there are actual polar mapstation subdirectories, this
program will be updated.

5 Input
 The following prompts are provided for input:

                    Scale: Enter the chart scale.

                    Zone: Enter the zone's area of coverage.

 Title for the plot: Enter the plot's title.

  Enter the side of
  the AOD (A OR B): Enter "A" or "B" to indicate which side of
                         the AOD is being used to store the data.

 Enter the subdirectory
      number to plot: Enter the number representing the appropriate
                         mapstation subdirectory.

 Do you want to draw
          1. the bounds of the subdirectory
          2. the individual segments
        Enter 1 or 2:

 Do you want to draw
              the land: [y] The default is "y" for yes.

 Do you want the political
            boundaries: [y] The default is "y" for yes.

3 PLOT_CDS_VWS
 Displays downsampled segments from ADRG CDs, with each segment
 represented by a rectangle. Downsampled segments from as many as
 8 ADRG CDs may be simultaneously displayed. Segments having full
 pixel counts (65536 pixels), or core segments, are drawn with
 black lines. Segments having lower pixel counts, or edge segments,
 are drawn with red lines along with a dissecting line running in
 one of the following directions:
                        vertically
                        horizontally
                        diagonally - from upper left to lower right
                        diagonally - from upper right to lower left

Dissecting lines are used to create a visual distinction between edge segments.

The entire ADRG CD must be downsampled before it can be displayed with PLOT_CDS_VWS.

4 Invocation
PLOT_CDS_VWS may only be executed on workstations running VAX/VWS software.   Hence, workstations SPARK1 and SPARK3 are available for program execution.   Perform the following step to execute PLOT_CDS_VWS:
    * Define a processing thread.
    * Type the command   RUN/NODEBUG V2_DIR:PLOT_CDS

4 Input
PLOT_CDS_VWS requires the following input.

    Enter the M4 zone to plot [0-4] (only one zone at a time can be
                            plotted).

    Enter the type of data to plot: 1 for full segments only
                                    2 for edge segments only
                                    3 for both (full and edge)

    Enter the last four characters of the chart name (DMA STOCK
    NUMBER).

Note: ADRG CD chart names are used as they utilize a geographic-based naming convention. For example, with JNC scale charts, chart X005 is located next to chart X006. MDFF CD numbers are not used as they are assigned sequentially at processing time. Hence, two adjacent charts, when not processed sequentially, possess non-sequential MDFF CD numbers. For example, JNC chart X005 has been assigned MDFF CD number 492 and JNC chart X006 has been assigned MDFF CD number 382 respectively).

Up to eight ADRG CDs may be displayed. When less than eight CDS are to be displayed, type <^Z> to terminate the last CD entry. PLOT_CDS_VWS will then display the segments.

Adjacent ADRG CDs (i.e., areas of contiguous coverage) have an overlap of edge segments. The edge segments that are shared by different CDs will have a pattern formed by the horizontal, vertical, or diagonal lines that are associated with the CDs.

3 PLOT_ODI
An interactive program that is used to define the area of coverage for a chart ODI build. Calcomp plots, depicting the total area of coverage, are created.

4 Invocation

E-29

PLOT_ODI may only be displayed on VAXstations that are running
DECwindows software. Currently, SPARK2 is the only available
DECwindows node.

Perform the following steps to execute PLOT_ODI:

* Set the appropriate processing thread
  $ADRGLOGS ###
  where ### is the three-character processing thread name

* Invoke PLOT_ODI
  $RUN/NODEB MDFFEXE:PLOT_ODI

4 Input_Output
Upon execution, the main selection box, containing three buttons
is displayed. The buttons are labeled PLOT, CONTROL, and QUIT.
Currently, only the PLOT and QUIT buttons are implemented.

   Select the PLOT option to continue the program.
   Select the QUIT option to terminate the program.

When the PLOT option is selected, a box containing all of the
available scales is displayed. Select the appropriate scale by
placing the arrow on the desired scale and clicking the first
mouse button.

The complete available area of coverage is represented by a grid
of colored squares. Each square within the grid represents an ADRG
CD-ROM. A CD-ROM (i.e., area of coverage) is selected for
inclusion (to the chart ODI build) by placing the arrow inside a
square and double clicking the first mouse button. Once it has
been selected, the square turns blue. Up to 200 CD-ROMS may be
selected.

A CD-ROM may be unselected by double clicking the first mouse
button inside a blue square. The square will then turn back to
it's original color (either red or black). Squares that are
colored red represent a CD-ROM that has already been incorporated
into a CAC. Black squares represent CD-ROMs that have not been
incorporated.

Once all of the desired CD-ROMs have been selected, the third
mouse button should be clicked. Another selection box, containing
PLOT and CANCEL options is displayed.

   Use the PLOT option when the selections are complete.
   (Additional CD-ROM's may be selected at a later date.)

   Use the CANCEL option to erase the CD-ROM selections that were
   made during this session.

When the PLOT option has been selected, a CalComp plot file that

E-30

shows the chart ODI area of coverage, is created. PLOT_ODI will
prompt for the following information

a title for the plot,
a filename for the plot file
a target completion date for the chart ODI

A file extension must not be included as part of the plot file
name. A directory path can be included as part of the file name.
Once the plot file is created and written, the main selection box
is again displayed.

4 Remote_Execution
PLOT_ODI requires the use of DECwindow software. Hence, only
those nodes, that have DECwindow software, can be used for display
(i.e., acting as the client).

PLOT_ODI can executed on any node as long as the client is located
on a DECwindows supported node. Utilizing the speed of faster
CPUs (e.g., node HARIER) enables faster program execution.

The following steps enable PLOT_ODI execution on node HARIER and
display on the SPARK2 monitor:

1) Add the following lines to your LOGIN.COM file:

```
$! Identify SPARK2 monitor as the display device.
$   define DECW$DISPLAY WSA1
$!
$! Identify the display device on node SPARK2
$   set display/create/node=SPARK2
```

2) Add the appropriate node and user name to the security
customization of your DECwindows session:

* Using the SESSION MANAGER menu, select the CUSTOMIZATION
option.

* Using the CUSTOMIZATION menu:

- Select the SECURITY option.

- Identify a node by placing the arrow inside of the node
name box and clicking the left mouse button. Using the
keyboard, type the name of the node that will need access
to the client (i.e., the SPARK2 monitor).

- Identify a user name by placing the arrow inside of the
user name box and clicking the left mouse button. Using
the keyboard, type the name of the user who will
need access to the client (i.e., the SPARK2 monitor).

E-31

- Once the appropriate names have been entered, select the ADD option.

The node and user names will now appear in the authorized user list. Additional names may be added to the authorization list by using the ADD option and repeating the above steps.

2 DOWNSAMPLE_CD

This program will produce the source ADRG CD-ROM's header file (the DMA ADRG product specification provides complete information concerning ADRG data). For each distribution rectangle on the ADRG CD-ROM, this program creates the DR Header File, Source Graphics Header File, and the DR coverage file. ADRG data are transformed from ARC projection to TS projection. Data are compressed by a factor of 4:1 due to the change in resolution from 256 pixels per inch (ARC) to 128 pixels per inch (TS).

2 DS_SEGS
This program is used to downsample specific Tessellated Spheroid (TS) segments from ADRG data that are stored on CD-ROM.

3 Input_Output
Perform the following steps to execute DS_SEGS:

A. Define a processing thread.

B. After placing the appropriate ADRG CD-ROM onto a CDREADER, mount the CD-ROM using the MOUNT_CDROM command.

C. Invoke DS_SEGS using the command: RUN/NODEB V2_DIR:DS_SEGS

D. DS_SEGS will prompt for the following information:

* The CDREADER that contains the ADRG CD-ROM.

* The path to use for writing downsampled TS data (the user is given 3 choices,
1) MDFF_SCRATCH:[CB_REPAIR]

2) CHART_SEGMENTS:

3) A user-defined area (see example 1 below),

* The M4 zone from which to downsample.

* The start and end row and col of TS segments to downsample.

DS_SEGS displays the number of Distribution Rectangles (DRs) on the ADRG CD-ROM and the number of Zone Distribution Rectangles (ZDR) associated with each DR.

DS_SEGS attempts to downsample each segment from all ZDRs associated with each DR. If a segment is not within a ZDR, a message is printed and the program begins downsampling for the next segment.

Once all ZDRs are exhausted for a particular DR, DS_SEGS tries downsampling the next DR and its ZDRs. When ADRG data are downsampled into a segment, a pixel count is displayed. A full or core segment must have pixel counts that add up to 65536 pixels. Unfilled or edge segments possess fewer than 65536 pixels and require additional downampled data from appropriate adjoining ADRG CD(s). Because a segment may be downsampled partially or wholly from more than one ZDR and DR, an individual segments' pixel count may add up to more than 65536 pixels.

The following are two example runs of DS_SEGS. EXAMPLE 1 downsamples one TS segment from an ADRG CD (row 1, column 125) and sends output to a user-defined area. EXAMPLE 2 downsamples two TS segments from an ADRG CD (row -22, columns 12 and 13) and sends the output to the MDFF_SCRATCH:[CB_REPAIR] area.

3 Examples
The following are examples of program execution.

Example 1: Downsample one TS segment from an ADRG CD (row 1, column 125) and send output to a user-defined area.

Example 2: Downsample two TS segments from an ADRG CD (row -22, columns 12 and 13) and send output to the MDFF_SCRATCH:[CB_REPAIR] area.

4   Example_1
    Downsample one TS segment from an ADRG CD (row 1,column 125) and send output to a user-defined area.

```
$ @mdffexe:use_adrg_logicals a5b
$ @mdffexe:mount_cdrom cdrom1
Mounting CDROM1
%I-ALLOC, _OSPREY$DUB0: allocated
%I-MOUNTED, ARC1_JNCXX05 mounted on _OSPREY$DUB0:

$ run/nodebug v2_dir:ds_segs
Enter CD device drive: cdrom1
Enter downsample segments directory
Enter: 1 for MDFF_SCRATCH:[CB_REPAIR]
       2 for CHART_SEGMENTS:
       3 for other: 3
Enter output destination (INCLUDE []):
```

```
        mdff$disk:[riedlinger.ds_segments]
        Enter M4 zone # to process (0/4): 2
        Enter start, end ROW of segments to downsample: 1,1
        Enter start, end COL of segments to downsample: 125,125

        There are 1 DR(s) on this CDROM
        Processing DR: 01
              Num ZDRs:              1
              ARC zone:              1
              ZDR num:               1
              Processing ROW/COL:              1              125
              PIXELS DOWNSAMPLED:           65536
        Enter M4 zone # to process (0/4): <CRTL_Z>
        $ @mdffexe:dismount_cdrom cdrom1
        dismountcd cdrom1
        Dismounting CDROM1


    4 Example_2
      Downsample two TS segments from an ADRG CD (row -22,
      columns 12 and 13) and send the output to the
      MDFF_SCRATCH:[CB_REPAIR] area.

        $ @mdffexe:mount_cdrom cdrom1
        Mounting CDROM1
        %I-ALLOC, _OSPREY$DUB0: allocated
        %I-MOUNTED, ARC1_JNCXX11 mounted on _OSPREY$DUB0:

        $ run/nodebug v2_dir:ds_segs
        Enter CD device drive: cdrom1
        Enter downsample segments directory
        Enter: 1 for MDFF_SCRATCH:[CB_REPAIR]
               2 for CHART_SEGMENTS:
               3 for other: 1
        Enter M4 zone # to process (0/4): 0
        Enter start, end ROW of segments to downsample: -22,-22
        Enter start, end COL of segments to downsample: 12,13

        There are 1 DR(s) on this CDROM
        Processing DR: 01
              Num ZDRs:              5
              ARC zone:             11
              ZDR num:               1
    TOP ZONE:              1
    BOT ZONE:              1
    Input zone of          0 not within ARC1 zone of              11
              ARC zone:             12
              ZDR num:               2
              Processing ROW/COL:             -22              12
              Segment not in bounds of this ZDR
              Processing ROW/COL:             -22              13
              Segment not in bounds of this ZDR
```

E-34

```
                ARC zone:              13
                ZDR num:               3
                Processing ROW/COL:              -22           12
                Segment not in bounds of this ZDR
                Processing ROW/COL:              -22           13
                Segment not in bounds of this ZDR
                ARC zone:              14
                ZDR num:               4
                Processing ROW/COL:              -22           12
                PIXELS DOWNSAMPLED:         65536
                Processing ROW/COL:              -22           13
                PIXELS DOWNSAMPLED:         65536
                ARC zone:              15
                ZDR num:               5
                Processing ROW/COL:              -22           12
                Segment not in bounds of this ZDR
                Processing ROW/COL:              -22           13
                Segment not in bounds of this ZDR
           Enter M4 zone # to process (0/4): <CTRL_Z>

           $ @mdffexe:dismount_cdrom cdrom1
           $ cdrom1
           Dismounting CDROM1
```

2 DUMP_CODEBOOK_STATUS
  DUMP_CODEBOOK_STATUS lists information about the codebooks of a
  specific CD or PA, that is being processed under a given
  processing thread.

  DUMP_CODEBOOK_STATUS is invoked through a DCL-like command. It is
  a "foreign" command that is defined in MDFFEXE:MDFF_SYMBOLS.COM.

  Format

  DUMP_CODEBOOK_STATUS [processing thread]

3 /CD_NUMBER
  /CD_NUMBER[=(CD number)[,...])]

  Dump specified MDFF CD number(s) codebooks.

3 /OUTPUT
  /OUTPUT=SYS$OUTPUT (default)
  /OUTPUT[=file specification]

  Directs the output to the specified file.

3 /PA_NUMBER
  /PA_NUMBER[=(PA number)[,...])]

Dump codebooks from specified PA Directories.

3 /PRINT
  /PRINT=MDFF_PRINT_QUEUE (default)
  /PRINT[=queue]

  Prints the output to a specified printer.

2 DUMP_COLPAL_DB
  Lists information about each existing color palette onto the
  user's video monitor, or SYS$OUTPUT.

  Type the following command to run DUMP_COLPAL_DB:
       RUN/NODEB   V2_DIR:DUMP_COLPAL_DB

3 Information_Record
 The following information is included for each color palette:

PA   PA   Left   Right   Top   Bot
ID   MS   MZ   Area   Date:Time   Long   Long   Lat   Lat

  Where
       PA ID: Palette identification number.
          MS: Map scale.
          MZ: Model 4 TS zone number.
     PA Area: Palette area.
   Date:Time: Date and time of color palette creation.
   Left Long: Left longitude boundary for area of coverage by this
              color palette.
  Right Long: Right longitude boundary for area of coverage by this
              color palette.
     Top Lat: Top latitude boundary for area of coverage by this
              color palette.
     Bot Lat: Bottom latitude boundary for area of coverage by this
              color palette.

2 DUMP_DS_SEGMENTS
 DUMP_DS_SEGMENTS lists information about the downsampled segments
 of a CD that is being processed under a given processing thread
 name.

 DUMP_DS_SEGMENTS is invoked through a DCL-like command. It is a
 "foreign" command that is defined in MDFFEXE:MDFF_SYMBOLS.COM.

 Format:

 DUMP_DS_SEGMENTS [processing thread]

3 /CD_NUMBER

```
/CD_NUMBER[=(CD number)[,...])]
```

Dumps the status of the downsampled segments of the
specified MDFF CD number(s).

3 /OUTPUT
/OUTPUT=SYS$OUTPUT (default)
/OUTPUT[=file specification]

Directs the output to the specified file.
By default, output is directed to SYS$OUTPUT.

3 /PRINT
/PRINT=MDFF_PRINT_QUEUE (default)
/PRINT[=queue]

Prints the output to the specified printer.
The default the print queue MDFF_PRINT_QUEUE

2 ENCODEKEY
This program prompts for a segment's row and column numbers.
These numbers are encoded to a key that is displayed as output.
The key comprises part of the segment's file name:

key = [(2 * maxrow + 1) * (col + maxrow)] + row + maxrow + 1

where maxrow = 9000

2 FIND_CD_STATUS
FIND_CD_STATUS.COM checks all processing threads for the
UNAVAILABILITY of CD-ROM readers. For each processing thread,
the current CHART_STATUS file is opened, and the
CHART_STATUS_RECORD is read and searched for each entry
containing the following information:

LAST_COMPLETED_PHASE that is less than or equal to ADRG_READ
PROCESSING_STATE   that is less than or equal to
                       DOWNSAMPLING_ADRG

When an entry is found to satisfy the above criteria, the
following message is displayed:
            Reader CDROM0#(ADRG) is not available!!!
            where
                    CDROM0# is the CD-ROM reader number
                    ADRG is the DMA ADRG CD name

Additionally, a call to SHOW_CDREADER_STATUS is included to show
which CD-ROM readers are currently in use.

FIND_CD_STATUS.COM relies on the following programs:

E-37

FIND_CD_STATUS_DRIVER - Determines current processing thread and
                        the number of entries contained in
                        the CHART_STATUS file
                        (which is passed to FIND_CD_STATUS).

        FIND_CD_STATUS - Receives number of entries for a
                         particular processing thread, and reads
                         the CHART_STATUS_RECORD searching for
                         entries, whose last_completed_phase and
                         processing_state meet the above criteria
                         (a message is displayed for those entries
                         meeting these criteria).

2 LIST_CHART_STATUS
 LIST_CHART_STATUS reads the CHART_STATUS file of the current or
 specified processing thread. Information about the thread and its
 associated charts are listed.

 LIST_CHART_STATUS is invoked through a DCL-like command. It is a
 "foreign" command in MDFFEXE:MDFF_SYMBOLS.COM and can be
 abbreviated as LIST.

 Format

     LIST [processing thread] qualifier(s)

3 Examples
 Example 1

   $ LIST

   Lists all entries in CHART_STATUS for the currently defined
   processing thread. If no thread has been set, the user will be
   asked to enter a valid thread name.

 Example 2

   $ LIST A3B

   Lists all entries in CHART_STATUS for processing thread A3B.

3 /ALL
 Lists every entry in the CHART_STATUS file. The program default.

 Format

     LIST /ALL

E-38

3 /CD_NUMBER
Lists entries in CHART_STATUS that possess the specified MDFF CD
number(s).

Format

    LIST /CD_NUMBER[=(CD number)[,...])]

4 Example
  $ LIST/CD_NUMBER=(511,600)

Lists CHART_STATUS entries with CD numbers 511 and 600.

3 /ENTRY
Lists specified entry number(s) in CHART_STATUS.

Format

    LIST /ENTRY[=(entry number)[,...])]

3 /FULL
Lists the complete status record. The default is NOFULL.

Format

    LIST /[NO]FULL (default)

3 /HEADER
Includes the header record in the listing. The default is
NOHEADER

Format

    LIST /[NO]HEADER

4 /HEADER=ONLY

Lists only the header record.

3 /OUTPUT
Directs the output to the specified file. By default, output is
sent to SYS$OUTPUT.

Format

    LIST /OUTPUT[=file specification]

4 Example

$ LIST A3B/PRINT=HARIER$LZ1/OUTPUT=MYFILE.OUT

   Lists all entries of processing thread A3B to an output file
named MYFILE.OUT and sends a copy of to the print queue
HARIER$LZ1 to be printed.

3 /PASS
Lists every entry in the CHART_STATUS file associated with the
specified PASS. By default, entries associated with PASS1 are
listed.

 Format

     LIST /PASS[=pass number]

3 /PRINT
Prints the listing on a specified printer. By default, output is
sent to the print queue that is defined by the DCL symbol
MDFF_PRINT_QUEUE.

 Format

     LIST /PRINT[=queue]

4 Example
 $ LIST A3B/PRINT=HARIER$LZ1/OUTPUT=MYFILE.OUT

   Lists all entries of processing thread A3B to an output file
named MYFILE.OUT and sends a copy to the print queue
HARIER$LZ1 to be printed.

4 /COPIES
   Indicates the number of copies to be printed.

 Format

     LIST /PRINT/COPIES=#

   where # is the number of copies to be printed on SYS$OUTPUT.

3 /STOCK_NUMBER
Lists CHART_STATUS entries that contain the specified DMA
stock number(s).

 Format

     LIST /STOCK_NUMBER[=("stock number")[,...])]

4 Example

$ LIST A3B/STOCK_NUMBER=TPCXX/HEADER

    List all CHART_STATUS entries with the substring 'TPCXX' as part
    of their DMA STOCK NUMBER. The header will also be displayed on
    the screen.

2 PROCESS_CHART
    Exists as a continuing background process that monitors and
    maintains CHART ODI processing.

2 READ_ADRG_DATA
    This program reads a specific file or a set of files
    (encompassing the CD-ROM image to be used in the chart ODI
    build) from CD-ROM or magnetic tape. The file(s) will be
    placed in the directory that is specified in a passed parameter
    to speed up overall MDFF processing.

2 SHOW_CDREADER_STATUS
    Displays the status of CD-ROM readers; availability and owner
    process names (for unavailable readers).

2 TEK
Prints files with extensions .TEK or .IMAGE on the TEKTRONIX
printer, via the TEK$PRINT queue. TEK is invoked through a DCL-
like command: it is a "foreign" command in
MDFFEXE:MDFF_SYMBOLS.COM.

Format
        TEK [/qualifiers] filename
    where
        qualifiers include [COPIES]
                      [CREATE]**
                      [KEEP]
                      [NONOTIFY]
                      [PRINT]
                      [RECOVER]**
                      [REMOVE]
                      [SHOW]

        filename is the .TEK or .IMAGE file name

** See sub-topic System_management_tools for information about
    these qualifiers.

3 /COPIES
Specifies the number of copies to be printed.

    /COPIES[=# of copies]   (default = 1)

4 Examples

    1. $TEK/PRINT=EXAMPLE.IMAGE/COPIES=2/KEEP=NONE/NONOTIFY

The TEK command prints two copies of the file 'EXAMPLE.IMAGE', deletes the .IMAGE file and the .TEK file upon printing, and does not notify the user when the job has completed printing.

3 /KEEP
Specifies which files should be deleted or kept after printing. The default is to keep the .IMAGE file and to delete the .TEK file.

The following values may be used with /KEEP for the following actions:

/KEEP (default = /KEEP=IMAGE)

/KEEP=NONE   .TEK and .IMAGE files are deleted.

/KEEP=ALL   .TEK and .IMAGE files are kept.

/KEEP=TEK   .TEK file is kept and the .IMAGE file is kept (if one exists).

/KEEP=IMAGE   .IMAGE file is kept and the .TEK file is deleted. (Default)

4 Examples
1. $TEK/PRINT=EXAMPLE.IMAGE/COPIES=2/KEEP=NONE/NONOTIFY

The TEK command prints two copies of the file 'EXAMPLE.IMAGE', deletes the .IMAGE file and the .TEK file upon printing and does not notify the user when the job has completed printing.

2. $TEK/PRINT=EXAMPLE.TEK/KEEP=TEK

The TEK command prints one copy of the .TEK file 'EXAMPLE.TEK' and does not delete the .TEK file after printing.

3 /NONOTIFY
The user will not be notified when the print job(s) have completed. The default is NOTIFY.

4 Examples
1. $TEK/PRINT=EXAMPLE.IMAGE/COPIES=2/NONOTIFY

The TEK command prints two copies of the file 'EXAMPLE.IMAGE' and does not notify the user when the job has completed printing.

3 /PRINT
Prints a .TEK or an .IMAGE file. Only file names with .TEK or .IMAGE extensions may be specified. Since there is no default file type, a file name using one of these extensions must be specified.

E-42

4 Examples
  1. $TEK/PRINT=EXAMPLE.TEK/KEEP=TEK

  The TEK command prints one copy of the .TEK file 'EXAMPLE.TEK'
  and does not delete the .TEK file after printing.

  2. $TEK/PRINT=EXAMPLE.IMAGE/COPIES=2/KEEP=NONE/NONOTIFY

  The TEK command prints two copies of the file 'EXAMPLE.IMAGE',
  deletes the .IMAGE file and the .TEK file upon printing, and does
  not notify the user when the job has completed printing.

3 /REMOVE
  Removes specified entry number(s) from TEK queue. Special
  privileges are required to remove job entries belonging to other
  users.

    /REMOVE={entry_num ¦ (entry_num,entry_num,...)
                       ¦ entry_num:entry_num}

4 Examples
  1. $TEK/REMOVE=(1,6,7)

  The TEK command identifies which job entries to remove from the
  FIFO queue: job entries 1, 6, and 7 will be removed from the
  queue.

  2. $TEK/REMOVE=1:5

  The TEK command uses a range of job entries to remove jobs from
  the FIFO queue: job entries 1 through 5 will be removed from the
  queue.

3 /SHOW
  Displays current status of jobs in the FIFO queue. The FIFO queue

  holds the following information:

    FIFO queue status
    IIS_IMAGES disk space
    TEK$PRINT queue status
    Jobs being printed

    Job states:
      WAITING   - waiting in the queue.
      PROCESSING - being converted from an .IMAGE file to a .TEK file.

      PROC_COMP - conversion complete, waiting to be printed.
      PRINTING   - being printed on the TEKTRONIX printer.
      COMPLETE   - job complete.

E-43

4 Examples

1. $TEK/SHOW

The TEK command shows information about the FIFO queue: all print
jobs and their status.

3 IMAGE_Files
When a file with an .IMAGE extension is specified, the print job
is placed into the FIFO queue In order to be printed, an .IMAGE
file must first converted into a .TEK file. The .TEK file will be
created in the user's IIS_IMAGES:[user] directory. Once it has
been created, the .TEK file is sent to the TEK$PRINT queue for
printing.

The status of a print job, which is still in the FIFO queue, is
provided through the TEK/SHOW command.

A print job can be removed from the FIFO queue with the TEK/REMOVE
command. TEK/REMOVE can not remove jobs in the TEK$PRINT queue.

By default, the .TEK file is deleted after printing. Use the
/KEEP=TEK qualifier to prevent the TEK file from being deleted
(see subtopic /KEEP).

3 Print_Queues
TEK relys on two queues for printing; a FIFO queue and the VAX/VMS
print queue TEK$PRINT.

4 FIFO_Queue
The FIFO queue is designed to hold print jobs and provide
information about their various states. All jobs are eventually
sent to the TEK$PRINT queue for printing. Available information
includes:

 FIFO queue status
 IIS_IMAGES disk space
 TEK$PRINT queue status
 Jobs being printed

 Print job states:
  WAITING   - waiting in the queue.
  PROCESSING - being converted from an .IMAGE file to a .TEK file.

  PROC_COMP - conversion complete, waiting to be printed.
  PRINTING   - being printed on the TEKTRONIX printer.
  COMPLETE   - job complete.

This information is available through the /SHOW qualifier (see
/SHOW).

TEK_CONTROL is an independent batch process that monitors the FIFO

E-44

queue and acts upon jobs according to their status. For example,
jobs that possess a "PROC_COMP" status are sent to the TEK$PRINT
queue for printing.

4 TEK$PRINT_Queue
A VAX/VMS print queue that prints .TEK files on the TEKTRONIX
printer.Information about this queue is available through
conventional VMS/DCL commands (see VMS HELP topic SHOW).

3 Subroutine_Invocation
TEK functionality is available through use of the C language
subroutine ADD_TO_TEK_QUEUE. All values are passed by reference.

Invocation syntax:

ADD_TO_TEK_QUEUE (print_filename, num_of_copies,
                  keep_flag, notify_flag);
  where
    print_filename = name of the .IMAGE or .TEK file to be printed

    num_of_copies = number of copies to be printed.

        keep_flag = 0, 1, 2, or 3   0 = /KEEP=NONE
                                           1 = /KEEP=IMAGE
                                           2 = /KEEP=ALL
                                           3 = /KEEP=TEK

      notify_flag = TRUE or FALSE   TRUE = NOTIFY
                                       FALSE = NONOTIFY

3 System_Management_Tools
 The following options are available for system management
 functions. System privileges are required for their use.

   /CREATE   Creates the TEK Job Queue data file.

   /RECOVER   Recovers the TEK Job Queue data file after a system
               failure has occurred.

3 TEK_Files
 Files with a .TEK extension are sent directly to the TEKTRONIX
 printer (i.e., TEK$PRINT) for printing.

 By default, the .TEK file is deleted after printing. Use the
 /KEEP=TEK qualifier to prevent the TEK file from being deleted
 (see sub-topic /KEEP).

2 TRIM_ODI_STRUCT
 TRIM_ODI_STRUCT moves designated compressed and downsampled
 segments from a specified processing thread into a trimmed

E-45

directory that is located on the same disk. In effect, moving
these segments out side of the area of coverage, "trims" the data
that is used as part of an ODI build. All trimmed segments, both
compressed and downsampled, are marked as being "TRIMMED" in the
SEGMENTS.DAT file.

Two types of bitmaps are used in performing the ODI trim:
    * Bitmaps for defining the current area of coverage
    * Bitmaps for the desired ODI bounds

Using the SEGMENTS.DAT file, bitmaps of the current area of
coverage, are built. Bitmaps for the ODI bounds are based upon
the ODI_BOUNDS.DAT file, where the southwest latitude/longitude
and northeast latitude/longitude corners of rectangles are used to
define the desired areas to keep (i.e., the bitmaps are composed
of these rectangles). For additional information about bitmaps,
see MDFFHELP topic BITMAPS.

Once built, the coverage and ODI bounds bitmaps are compared to
determine which segments are to be trimmed (i.e., moved to the
trim directory); all segments that lie outside of these rectangles
will be trimmed.

3 Input_Output
 There is no user input. Necessary information for performing the
 ODI trim is automatically obtained from the SEGMENTS.DAT and
 ODI_BOUNDS.DAT files. The ODI_BOUNDS.DAT file can be modified
 through the PASS2 phase of CHART_CONTROL.

 TRIM_ODI_STRUCT moves the segments to a scratch area instead of
 permanently deleting them. This is accomplished by creating a
 trimmed directory on the same disk and level as the parent
 directory of the map data.

 EXAMPLE 1: Trimmed Directory for Compressed Segments.
 If the current parent directory is CHART_ODI:[MDFF.A5B],
 then a new trimmed directory named CHART_ODI:[MDFF.A5B_TRIMMED]
 is created.

 EXAMPLE 2: Trimmed Directory for Downsampled Segments.
 If the current parent directory is CHART_SEGS:[MDFF.A5B],
 then a new trimmed directory named CHART_SEGS:[MDFF.A5B_TRIMMED]
 is created.

 Once the trimmed directories have been created, palette
 directories and row directories are created within the trimmed
 directories, to house the segments that are to be trimmed (i.e.,
 moved into these new areas).

3 Invocation
 TRIM_ODI_STRUCT was designed to be executed as a detached process
 from CHART_CONTROL but can also be executed from VMS.

Use the following instructions to execute from the VMS level:

    (1)   Set the process thread of the map data to be trimmed.
    (2)   Type RUN/NODEB MDFFEXE:TRIM_ODI_STRUCT.

TOPIC FILE: CAC_SOURCE_CODE.HLP

The following text and subtopics appear when **CAC_Source_Code** is selected as an **MDFFHELP** subtopic:

```
CAC_Source_Code

  This topic provides software descriptions such as how to
  call a particular subprogram and what setup information is
  required (e.g. include files, variable declarations,
  etc.).

  The programming languages used are included as file
  extensions:  routines written in FORTRAN have a file
  extension ".FOR" and routines written in C have a file
  extension ".C".

  Additional information available:

  Overview            BITMAP_APPL.C            BITMAP_ODI.C
  BITMAP_PROCS.C      BITMAP_SEGMENTSDAT.C     BITMAP_SOURCE.C
  BITMAP_TRANS.C      BUILD_CD_ID.C      CAC.C    CAC_MISC.C
  CAC_UTIL.C          CCOMMA.C           CHECK_LON_ORIENTATION.C
  CLEAN_UP.C COORDINATES.FOR        CREATE_PA_DIR.C
  DECIMAL.FOR         DTR_PROCS.FOR
  FCOMMA.C    FILE_ATTR_PROCS.C      FIND_CAC_BITMAP.C
  FIND_FILE.FOR       GENERIC_QUEUE_STOPPED.FOR
  GET_DS_SEGMENT_NAME.FOR          GET_ODI_DATA.C
  GET_PA_SEGMENT_NAME.FOR          GET_PID.FOR
  GET_UNIX_BINARY_TIME.C           IIS_PLOT_PROCS.FOR
  LL_MAXMIN.C         MAP_DIR_PROCS.C QAL_MAXMIN.C
  QUAD_PROCS.C        RC_MAXMIN.C     SEND_MAIL.FOR
  SEND_MAIL_C.C       TIMEM.MAR       USE_ADRG_LOGICALS.FOR
```

The following text comprises this **MDFFHELP** topic file. Note that subtopics begin with the key "2", which is located in column 1.

1 CAC_Source_Code

    This topic provides software descriptions such as how to
    call a particular subprogram and what setup information is
    required (e.g., include files, variable declarations, etc.).

    The programming languages used are included as file extensions:
    routines written in FORTRAN have a file extension ".FOR" and
    routines written in C have a file extension ".C".

F-5

2 Overview

Available functions are briefly described. The programming languages used are included as file extensions: routines written in FORTRAN have a file extension ".FOR" and routines written in C have a file extension ".C".

BITMAP_APPL.C: A suite of high-level routines that perform a variety of complex bitmap operations.

BITMAP_ODI.C: A suite of routines that create bitmaps from the MDFF:SCRATCH:[000000]ODI_BOUNDS.DAT file.

BITMAP_PROCS.C: A suite of low-level routines that perform a variety of basic bitmap operations.

BITMAP_SEGMENTSDAT.C: A suite of routines that build bitmaps from the SEGMENTS.DAT file.

BITMAP_SOURCE.C: A suite of routines that build bitmaps from downsampled and compressed segment data.

BITMAP_TRANS.C: Combines two or more bitmaps.

BUILD_CD_ID.C: Creates the [ID]CD_ID file according to the format of table 10 HTI specification.

CAC.C: A suite of routines that provide portable support for reading CAC CD-ROMs.

CAC_MISC.C: A suite of low-level CAC reader routines.

CAC_UTIL.C: A suite of high-level CAC reader routines.

CCOMMA.C: Places a comma(s) in an integer number. Can only be used with other C routines.

CHECK_LON_ORIENTATION.C: Determines whether a minimum longitude and a maximum longitude are indeed the true minimum and maximum longitude values for a data set.

CLEAN_UP.C: Transverses the current processing thread directory structure and performs functions necessary for trimming the chart ODI.

COORDINATES.FOR: Converts latitude/longitude coordinates into their degrees, minutes, and seconds equivalents.

CREATE_PA_DIR.C: --- TO BE ADDED LATER ---

DECIMAL.FOR: Converts a coordinate (in character format) to its decimal equivalent. Designed to work with the routine COORDINATES.FOR

DTR_PROCS.FOR: A suite of routines for converting to/from Datatrieve data storage format.

FCOMMA.FOR: Places a comma(s) in an integer number. Can only be used with other FORTRAN routines.

FILE_ATTR_PROCS.C: A suite of routines that provide information about VAX/VMS files and directories.

FIND_CAC_BITMAP.C: Returns all CAC logged bitmap file names that are stored in the MDFF_SYSTEM:[BITMAPS] directory for a given scale and zone.

FIND_FILE.FOR: Uses search and file names to find a file.

GENERIC_QUEUE_STOPPED.FOR: Determines whether a generic queue is stopped.

GET_DS_SEGMENT_NAME.FOR: Builds the path name for a downsampled segment file.

GET_ODI_DATA.C: Reads mapstation subdirectories, extracts row/column values and returns the sorted (by ascending row) values.

GET_PA_SEGMENT_NAME.FOR: Builds the path name for a compressed segment file.

GET_PID.FOR: A suite of routines that obtain a Process Identification (PID) code.

GET_UNIX_BINARY_TIME.C: A suite of routines that convert a VAX time string into the corresponding UNIX binary time.

IIS_PLOT_PROCS.FOR: A suite of routines that enable the use of graphics device and generation of hardcopy plots on the TEKTRONIX printer.

LL_MAXMIN.C: A suite of routines that determine the true pair of minimum and maximum coordinates (Lat/Lon) from a set of coordinate pairs.

MAP_DIR_PROCS.C: A suite of routines that perform a variety of operations on a MAP directory and the files it contains.

F-7

QAL_MAXMIN.C: A suite of routines that use the ADRG QAL file
to determine the true minimum and maximum
latitude/longitude coordinates for polar
data.

QUAD_PROCS.C: A suite of routines that compute the minimum
and maximum latitude/longitude coordinates
for polar data.

RC_MAXMIN.C: A suite of routines that determine the true
pair of minimum and maximum row/column values
from a set of row/column pairs.

SEND_MAIL.FOR: Sends a VAX/VMS mail message from programs
written in FORTRAN programming language.

SEND_MAIL_C.C: Sends a VAX/VMS mail message from programs
written in C programming language.

TIMEM.MAR: A C language routine that separates one
quadword into two longwords.

USE_ADRG_LOGICALS.FOR: Inserts the specified processing thread's
logical name table into the LNM$FILE_DEV
logical name search path.

2 BITMAP_APPL.C

Contains a suite of high-level routines that perform a variety of
complex bitmap operations. For additional information about
bitmaps, see MDFFHELP main topic BITMAPS.

3 Include_Files

There are two files that must be included for use with
BITMAP_APPL.C. Their names and brief descriptions follow.

V2_DIR:BITMAP_APPL.H   Contains data definitions used by the
subroutines.

V2_DIR:DATA_DEFS.H   Contains type definitions for common
variables (e.g., scale, zone)

The following subroutines are required for linkage. For
additional information about each subroutine, see MDFFHELP under
the main topic CAC_SOURCE_CODE.

BITMAP_PROCS.C   A Suite of low-level routines that perform a
variety of basic bitmap operations.

LL_MAXMIN.C   A suite of routines that determine the true
pair of minimum and maximum latitude/longitude
coordinates.

QUAD_PROCS.C A suite of routines that compute the minimum and maximum latitude/longitude coordinates for polar data.

RC_MAXMIN.C A suite of routines that determine the true pair of minimum and maximum row/column values from a set of row/column pairs.

3 Suite_Description
BITMAP_APPL.C contains the following subroutines:

4 BITMAP_APPL_INIT
This function is passed the chart scale of the data being manipulated and should be invoked first, before any other routine within this suite. The following syntax is used to invoke bitmap_appl_init:

  void bitmap_appl_init (SCALE *mapscale)

4 BITMAP_COUNT_INIT
Sets the count of bitmaps for all zones to zero. There are no values returned or arguments passed. The following syntax is used to invoke bitmap_count_init:

  void bitmap_count_init (void)

4 BITMAP_FROM_LATLON
When passed a scale, zone, and latitude/longitude coordinates of the lower left and upper right corner of a rectangle, this function builds a bitmap of this rectangle and returns the minimum and maximum row/column values of the bitmap. The following syntax is used to invoke bitmap_from_latlon:

  BIT *bitmap_from_latlon (SCALE *mapscale, ZONE *mapzone,
                           LON *minlon, LON *maxlon,
                           LAT *minlat, LAT *maxlat,
                           COL *mncol, COL *mxcol,
                           ROW *mnrow, ROW *mxrow)
where

BIT	The returned bitmap
mapscale	Contains the chart scale
mapzone	Contains the TS zone the data lie within
minlon	Contains the minimum segment longitude coordinate
maxlon	Contains the maximum segment longitude coordinate
minlat	Contains the minimum segment latitude coordinate
maxlat	Contains the maximum segment latitude coordinate
mncol	Contains the minimum segment column number
mxcol	Contains the maximum segment column number
mnrow	Contains the minimum segment row number
mxrow	Contains the maximum segment row number

F-9

4 COMBINE_BOXES_RET
Combines all rectangles defined by the functions
"define_boxes_send" and "define_boxes_per_zone" into one bitmap
for a given zone. The following syntax is used to invoke
combine_boxes_ret:

```
BIT *combine_boxes_ret (ZONE *zone)
```

where
```
     BIT   Is the returned bitmap
     zone  Contains the TS zone the data lie within
```

4 DEFINE_BOXES_PER_ZONE
Is passed a rectangle that is contained within one (and only one)
zone. The rectangle is defined by minimum and maximum
latitude/longitude coordinates. This function stores the
latitude/longitude and row/column coordinates in separate two-
dimensional arrays. The following syntax is used to invoke
define_boxes_per_zone:

```
void define_boxes_per_zone (ZONE *zone, LON *minlon, LON *maxlon,
                            LAT *minlat, LAT *maxlat)
```

where
```
        zone    Contains the TS zone the data lie within
      minlon    Contains the minimum segment longitude coordinate
      maxlon    Contains the maximum segment longitude coordinate
      minlat    Contains the minimum segment latitude coordinate
      maxlat    Contains the maximum segment latitude coordinate
```

4 DEFINE_BOXES_SEND
When passed in a rectangle defined by minimum and maximum
latitude/longitude coordinates, this function determines which
zones the rectangle covers. It creates little rectangles, for
each zone and stores their latitude/longitude and row/column
coordinates in separate two-dimensional arrays. The following
syntax is used to invoke define_boxes_send:

```
void define_boxes_send (LON *minlon, LON *maxlon,
                        LAT *minlat, LAT *maxlat)
```

where
```
      minlon    Contains the minimum segment longitude coordinate
      maxlon    Contains the maximum segment longitude coordinate
      minlat    Contains the minimum segment latitude coordinate
      maxlat    Contains the maximum segment latitude coordinate
```

4 GET_BITMAP_FROM_DISK
Prompts user for a bitmap prefix name of a bitmap located in the
MDFF_SYSTEM:[BITMAPS] directory, reads in this bitmap, and returns

it to the calling point. The following syntax is used to invoke
get_bitmap_from_disk:

```
   BIT *get_bitmap_from_disk (SCALE *mapscale, ZONE *mapzone,
                              char *filename, COL *mncol,
                              COL *mxcol, ROW *mnrow, ROW *mxrow)
```

where

BIT	The returned bitmap
mapscale	Contains the chart scale
mapzone	Contains the TS zone the data lie within
filename	Contains the bitmap file name
mncol	Contains the minimum segment column number
mxcol	Contains the maximum segment column number
mnrow	Contains the minimum segment row number
mxrow	Contains the maximum segment row number

4 GET_COORDS
Prompts user for the latitude/longitude coordinates of the lower
left corner and the upper right corner of a rectangle. The
following syntax is used to invoke get_coords:

```
   unsigned char get_coords (LON *minlon, LON *maxlon,
                             LAT *minlat, LAT *maxlat)
```

where

unsigned char	The return status value
minlon	Contains the minimum segment longitude coordinate
maxlon	Contains the maximum segment longitude coordinate
minlat	Contains the minimum segment latitude coordinate
maxlat	Contains the maximum segment latitude coordinate

4 MINMAX_INIT
Once minimum and maximum row/column values are determined, this
function should be invoked to pass these values to other
bitmap_appl routines. The following syntax is used to invoke
minmax_init:

```
   void minmax_init (ZONE *zone, COL *mncol, COL *mxcol,
                     ROW *mnrow, ROW *mxrow)
```

where

zone	Contains the TS zone the data lie within
mncol	Contains the minimum segment column number
mxcol	Contains the maximum segment column number
mnrow	Contains the minimum segment row number
mxrow	Contains the maximum segment row number

4 STORE_BOX_COORDINATES
Stores the row/column and latitude/longitude coordinates of the
lower left and upper right corners of a defined rectangle into
separate two-dimensional arrays. The following syntax is used to
invoke store_box_coordinates:

   void store_box_coordinates (ZONE zone)

where
      zone Contains the TS zone the data lie within

2 BITMAP_ODI.C
Contains a suite of C language routines that create bitmaps from
the MDFF_SCRATCH:[000000]ODI_BOUNDS.DAT file. The ODI_BOUNDS.DAT
file is stored in the MDFF_SCRATCH:[000000] directory for the
current processing thread. Subtopics include requirements and
individual routines that are contained within the suite.

3 Include_Files
There are two files that must be included for use with
BITMAP_ODI.C. Their names and a brief description follow.

   V2_DIR:BITMAP_ODI.H   Contains data definitions used by
                             subroutines within the suite.

   V2_DIR:DATA_DEFS.H   Contains type definition for common
                             variables (e.g., chart scale, zone)

The following subroutines are required for linkage. For
additional information about each subroutine, see MDFFHELP under
the main topic CAC_SOURCE_CODE.

       BITMAP_APPL.C   A suite of high-level routines that perform
                        complex bitmap application functions.

       BITMAP_PROCS.C   A suite of low-level routines that perform a
                        variety of basic bitmap operations.

       LL_MAXMIN.C   A suite of routines that determine the true
                        pair of minimum and maximum coordinates.

       ODI_BOUNDS.C   NOT YET ADDED TO MDFFHELP

       QUAD_PROCS.C   A suite of routines that compute the minimum
                        and maximum latitude/longitude coordinates for
                        polar data.

       RC_MAXMIN.C   A suite of routines that determine the true
                        pair of minimum and maximum row/column values
                        from a set of row/column pairs.

F-12

3 Suite_Description
BITMAP_ODI.C contains the following subroutines:

4 BUILD_ODI_BOUNDS_BITMAP
When passed a scale and zone, this function builds a bitmap using
the bounds from the ODI_BOUNDS.DAT file. The bitmap's
minimum/maximum row/column values are returned and are set to
accommodate the ODI bounds. The following syntax is used to invoke
build_odi_bounds_bitmap:

```
BIT *build_odi_bounds_bitmap (SCALE *mapscale, ZONE *mapzone,
                             COL *mncol, COL *mxcol, ROW *mnrow,
                             ROW *mxrow)
```

where

BIT	The returned bitmap
mapscale	Chart scale (passed)
mapzone	TS zone number (passed)
mncol,mxcol	minimum & maximum column numbers (returned)
mnrow,mxrow	minimum & maximum row numbers (returned)

5 Example
The following code segment provides an example of usage.

```
BIT *map;              /* The bitmap      */
SCALE mapscale;        /* chart scale     */
ZONE  mapzone;         /* TS zone number */
ROW   minrow, maxrow;  /* minimum & maximum row #s */
COL   mincol, maxcol;  /* minimum & maximum column #s */

mapscale = 5;
mapzone  = 4;

/* After this call, map will be a pointer to a new   */
/* bitmap that has been created bounds to house the  */
/* odi for the zone requested (zone 4, in this case).*/
/* The minimum/maximum row/column values of the      */
/*  bitmap (map) will be returned.                   */

map = build_odi_bounds_bitmap2 (&mapscale, &mapzone,
                               &mincol,
                               &maxcol,
                               &minrow,
                               &maxrow);
```

4 BUILD_ODI_BOUNDS_BITMAP2
When passed a scale and zone, this function builds a bitmap using
the bounds from the ODI_BOUNDS.DAT file. Additionally, these
bounds may be modified by the user. Alternate bounds, determined
by the user, can be used for the bitmap's minimum/maximum
row/column values and must be passed to the module BITMAP_APPL.C

F-13

(see BITMAP_APPL.C) prior to invoking this routine. The following
syntax is used to invoke build_odi_bounds_bitmap:

```
BIT *build_odi_bounds_bitmap2 (SCALE *mapscale, ZONE *mapzone)
```

where
```
        BIT    The returned bitmap
   mapscale    The chart scale (passed)
   mapzone     The TS zone number (passed)
```

5 Example
The following code segment provides an example of usage.

```
        BIT *map;         /* the bitmap       */
        SCALE mapscale;   /* chart scale      */
        ZONE  mapzone;    /* TS zone number */
        ROW   minrow, maxrow; /* minimum & maximum row #s */
        COL   mincol, maxcol; /* minimum & maximum column #s */

        mapscale = 5;
        mapzone  = 4;
```

```
/* These are the user-selected min/max row/col bitmap values */
        minrow   = 100;
        maxrow   = 125;
        mincol   = 150;
        maxcol   = 175;
```

```
/* The min/max values are passed to the routine minmax_init    */
/* (contained within the module BITMAP_APPL.C) to obtain a     */
/* value for mapzone.  mapzone is an argument used by the      */
/* build_odi_bounds_bitmap2 routine.                           */

    minmax_init (&mapzone, &mincol, &maxcol, &minrow, &maxrow);


        /* After this next call, map will be a pointer to the new */
        /* bitmap that has been created to house the odi bounds   */
        /* for the zone requested (zone 4, in this case).  Note,  */
        /* the minimum/maximum values of the bitmap (map) are the */
        /* same minimum/maximum values that were passed to the    */
        /* minmax_init routine. */


        map = build_odi_bounds_bitmap2 (&mapscale, &mapzone);
```

2 BITMAP_PROCS.C
Contains a suite of low-level C language routines that perform
a variety of basic bitmap operations. Subtopics include
requirements, individual routines within the suite, and an
example of usage.

3 Requirements

4 Include_Files
The file, "V2_DIR:BITMAP_PROCS.H", contains data definitions that are used by suite routines and must be included in the source code.

4 Global_Variables
Global variables, which must be declared external (i.e., "extern"), include the following:

    long maxrow, maxcol;   The maximum row and column values of
                                data in the bitmap.

    long minrow, mincol;   The minimum row and column values of
                                data in the bitmap.

        long total_bits;   Total number of bits in the bitmap.

        long num_rows;   Number of rows in the bitmap.

        long num_cols;   Number of cols in the bitmap.

3 Suite_Description
BITMAP_PROCS.C contains the following routines:

4 ADD_BUF_TO_BM
Adds two rows of clear bits around the bitmap. This is performed so that the routines used to traverse the bitmap do not have to check for bounds. The following syntax is used to invoke add_buf_to_bm:

    void add_buf_to_bm (void)

4 BITMAP_INIT
Should be invoked before a bitmap is created. This function adds a two-position wide buffer to the minimum and maximum values, determines the number of rows and columns the bitmap will need in order to hold the data, and computes the total bits needed to construct the bitmap. The following syntax is used to invoke bitmap_init:

    void bitmap_init (long *mnrow, long *mncol,
                      long *mxrow, long *mxcol)

where
    long maxrow, maxcol;   The maximum row and column values of
                                data in the bitmap.

    long minrow, mincol;   The minimum row and column values of
                                data in the bitmap.

F-15

4 CHECK_DOWN
Determines whether the bit, which is located below the bit passed
in, is set.   The following syntax is used to invoke check_down:

    long check_down (BIT map[], long *bitrow, long *bitcol)

    where
        long   Return status that will possess one of the
                            following values:
                                FALSE if the above bit is set
                                TRUE if the above bit is not set
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value
        long bitcol   The bitmap's column value

4 CHECK_DOWN_LEFT
Determines whether the bit, which is located below and to the left
of the bit passed in, is set.   The following syntax is used to
invoke check_down_left:

    long check_down_left (BIT map[], long *bitrow, long *bitcol)

    where
        long   Return status that will possess one of the
                            following values:
                                FALSE if the above bit is set
                                TRUE if the above bit is not set
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value
        long bitcol   The bitmap's column value

4 CHECK_DOWN_RIGHT
Determines whether the bit, which is located below and to the
right of the bit passed in, is set.   The following syntax is used
to invoke check_down_right:

    long <check_down_right (BIT map[], long *bitrow, long *bitcol)

    where
        long   Return status that will possess one of the
                            following values:
                                FALSE if the above bit is set
                                TRUE if the above bit is not set
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value
        long bitcol   The bitmap's column value

4 CHECK_LEFT
Determines whether the bit, which is located to the left of the
bit passed in, is set.   The following syntax is used to invoke
check_left:

```
long check_left (BIT map[], long *bitrow, long *bitcol)
```

where
   long   Return status that will possess one of the
                     following values:
                        FALSE if the above bit is set
                        TRUE if the above bit is not set
   BIT map[]   The bitmap
   long bitrow   The bitmap's row value
   long bitcol   The bitmap's column value

4 CHECK_RIGHT

Determines whether the bit, which is located to the right of the
bit passed in, is set. The following syntax is used to invoke
check_right:

```
long check_right (BIT map[], long *bitrow, long *bitcol)
```

where
   long   Return status that will possess one of the
                     following values:
                        FALSE if the above bit is set
                        TRUE if the above bit is not set
   BIT map[]   The bitmap
   long bitrow   The bitmap's row value
   long bitcol   The bitmap's column value

4 CHECK_UP

Determines whether the bit, which is located above the bit passed
in, is set. The following syntax is used to invoke check_up:

```
long check_up (BIT map[], long *bitrow, long *bitcol)
```

where
   long   Return status that will possess one of the
                     following values:
                        FALSE if the above bit is set
                        TRUE if the above bit is not set
   BIT map[]   The bitmap
   long bitrow   The bitmap's row value
   long bitcol   The bitmap's column value

4 CHECK_UP_LEFT

Determines whether the bit, which is located above and to the left
of the bit passed in, is set. The following syntax is used to
invoke check_up_left:

```
long check_up_left (BIT map[], long *bitrow, long *bitcol)
```

where
   long   Return status that will possess one of the
                     following values:

F-17

```
                              FALSE if the above bit is set
                              TRUE  if the above bit is not set
           BIT map[]        The bitmap
           long bitrow      The bitmap's row value
           long bitcol      The bitmap's column value
```

4 CHECK_UP_RIGHT

Determines whether the bit, which is located above and to the
right of the bit passed in, is set. The following syntax is used
to invoke check_up_right:

```
   long check_up_right (BIT map[], long *bitrow, long *bitcol)
```

   where
```
        long              Return status that will possess one of the
                          following values:
                             FALSE if the above bit is set
                             TRUE  if the above bit is not set
        BIT map[]         The bitmap
        long bitrow       The bitmap's row value
        long bitcol       The bitmap's column value
```

4 CLEAR_BIT

Clears a bit in the bitmap. The bit is determined by the bitmap's
row and column that are passed as arguments. The following syntax
is used to invoke clear_bit:

```
   void clear_bit (BIT map[], long *bitrow, long *bitcol)
```

   where
```
        BIT map[]        The bitmap
        long bitrow      The bitmap's row value
        long bitcol      The bitmap's column value
```

4 CLEAR_BITMAP

Clears every bit contained within the bitmap. The following
syntax is used to invoke clear_bitmap:

```
   void clear_bitmap (BIT map[])
```

   where
```
        BIT map[]        The bitmap
```

4 CLOSE_BITMAP_FILE

Closes a binary file that contains a bitmap. The following syntax
is used to invoke close_bitmap_file:

```
   void close_bitmap_file (long *filedesc)
```

   where
```
        long filedesc     Binary file descriptor
```

F-18

4 CONVERT_BM_TO_RC
Uses bitmap row and column values to determine the equivalent row
and column values of the original data. The following syntax is
used to invoke convert_bm_to_rc:

```
void convert_bm_to_rc (long *bitrow, long *bitcol,
                       long *row, long *col)
```

where

long bitrow	The bitmap's row value
long bitcol	The bitmap's column value
long row	The original data's row value
long col	The original data's column value

4 CONVERT_RC_TO_BM
Uses original data row and column values to determine the
equivalent bitmap row and column values. The following syntax is
used to invoke convert_rc_to_bm:

```
void convert_rc_to_bm (long *row, long *col,
                       long *bitrow, long *bitcol)
```
where

long row	The original data's row value
long col	The original data's column value
long bitrow	The bitmap's row value
long bitcol	The bitmap's column value

4 COPY_BITMAP
Takes in a source bitmap and copies it to a destination bitmap.
The following syntax is used to invoke copy_bitmap:

```
void copy_bitmap (BIT destination[], BIT source[])
```

where

| BIT destination[] | The destination bitmap |
| BIT source[] | The source bitmap |

4 CREATE_BITMAP
Returns a pointer to a bitmap by allocating the total number of
bits needed to contain the data. A function to clear the bitmap
is then called. The following syntax is used to invoke
create_bitmap:

```
BIT *create_bitmap (void)
```

4 DESTROY_BITMAP
Releases memory, that has been reserved for the bitmap, back to
the system. The following syntax is used to invoke destroy_bitmap:

```
void destroy_bitmap (BIT map[])
```

4 DUMP_BITMAP

Writes an ASCII representation of the bitmap into the specified
file. The following syntax is used to invoke dump_bitmap:

    void dump_bitmap (FILE *filedesc, BIT map[])

where
        FILE filedesc   The file specification
        BIT   map[]   The bitmap

4 INDEX
When passed a valid bitmap row and column, this function returns
an index to that row and column in the bitmap. The following
syntax is used to invoke index:

    long index (long *bitrow, long *bitcol)

where
        long   returns the index value
        long bitrow   bitmap row value
        long bitcol   bitmap column value

4 IS_CLEAR
Returns TRUE if the bit, as determined by the bitmap row and
column value passed in, is cleared. The following syntax is used
to invoke is_clear:

    long is_clear (BIT map[], long *bitrow, long *bitcol)

where
        long   Returns the status, contains one of
                          following values:
                              TRUE if the bit is clear
                              FALSE if the bit is not clear
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value
        long bitcol   The bitmap's column value

4 IS_SET
Returns TRUE if the bit, as determined by the bitmap row and
column values passed in, is set. The following syntax is used to
invoke is_set:

    long is_set (BIT map[], long *bitrow, long *bitcol)

where
        long   Returns the status, contains one of
                          following values:
                              TRUE if the bit is set
                              FALSE if the bit is not set
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value
        long bitcol   The bitmap's column value

4 MERGE_BITMAPS
This function merges two bitmaps. The merged bitmap, specified as the destination, is returned. The bitmaps must be of the same dimension. The following syntax is used to invoke merge_bitmap:

    void merge_bitmaps (BIT destination[], BIT source[])

  where
        BIT destination[]   One of the bitmaps to be merged and also
                            the destination bitmap
        BIT source[]   The other source bitmap to be merged

4 OPEN_BITMAP_FILE
Opens a binary file, in which a bitmap may be stored, and returns a file descriptor to the open file. The following syntax is used to invoke open_bitmap_file:

    long open_bitmap_file (char filename[])

  where
        long   Returned open file descriptor
        char filename   The name of the binary file

4 READ_IN_BITMAP
Reads a bitmap from the binary file that is defined by the file descriptor. Use the routine, open_bitmap_file, to open the binary file (see subtopic open_bitmap_file). The following syntax is used to invoke read_in_bitmap:

    BIT *read_in_bitmap (long *filedesc)

  where
        BIT map[]   The bitmap to be read
        long filedesc   Binary file descriptor

4 REMOVE_BUF_FROM_BM
Removes the two rows of clear bits (that were added by the function add_buf_to_bm) around the bitmap. The following syntax is used to invoke remove_buf_from_bm:

    void remove_buf_from_bm (void)

4 SET_BIT
Sets a bit in the bitmap. The bit is determined by the bitmap's row and column that are passed as arguments. The following syntax is used to invoke set_bit:

    void set_bit (BIT map[], long *bitrow, long *bitcol)

  where
        BIT map[]   The bitmap
        long bitrow   The bitmap's row value

```
        long bitcol    The bitmap's column value
```

4 SET_BITMAP
 Sets every bit contained within the bitmap. The following
 syntax is used to invoke set_bitmap:

```
   void set_bitmap (BIT map[])
```

  where
```
        BIT map[]       The bitmap
```

4 TOGGLE_BIT
 Toggles the state of a bit within the bitmap. The bit is
 determined by the bitmap's row and column that are passed as
 arguments. The following syntax is used to invoke toggle_bit:

```
   void toggle_bit (BIT map[], long *bitrow, long *bitcol)
```

  where
```
        BIT map[]       The bitmap
        long bitrow     The bitmap's row value
        long bitcol     The bitmap's column value
```

4 WRITE_OUT_BITMAP
 Writes a bitmap into the binary file that is defined by the file
 descriptor. Use the routine, open_bitmap_file, to open the binary
 file (see subtopic open_bitmap_file). The following syntax is
 used to invoke write_out_bitmap:

```
   void write_out_bitmap (long *filedesc, BIT map[])
```

  where
```
        long filedesc   Binary file descriptor
        BIT  map[]      The bitmap to be written
```

3 Example
 The following C language program provides examples of usage.

```
 #include "bitmap_procs.h"
 #include "v2_dir:map_dir_procs.h"

 BIT *bitmap;
 struct coord POINTS [MAX_SEG];

 main ()
 {
  long i;
  long count;            /* Number of Segments in a zone.   */
  long bitrow, bitcol;   /* Actual row/col index in bitmap. */
  FILE *fp;              /* File where ASCII bitmap will be */
                         /* dumped.                         */
  long filedesc;         /* File descriptor of bitmap       */
```

```
                        /* memory file.                      */
  long scale, zone;

  long mncol, mnrow, mxcol, mxrow; /* Min/Max values of data to */
                                   /* be placed in the bitmap.  */


  char ascii_filename[] = "bit.out"; /* ASCII Dump of bitmap.  */
  char memory_filename[] = "mem.out"; /* Memory Dump of bitmap. */

/**********************************************/
/* Enter Scale and Zone to be placed in bitmap */
/**********************************************/
  scale = 3;
  zone = 2;

/**********************************************/
/* Open the ASCII bitmap file.               */
/**********************************************/
  if ((fp = fopen (ascii_filename, "w")) == NULL)
  {
    printf ("ERROR:  Opening output file\n");
    exit (0);
  }

/**********************************************/
/* Place all segments in a array of records.  */
/**********************************************/
  count = 0;
  count = plot_segments (POINTS, zone);

/**********************************************/
/* Find Max/Min values for data.             */
/**********************************************/
  rc_maxmin_init (&scale, &zone);

  for (i=0;i<count;i++)
    rc_maxmin_send ((&(POINTS[i].row)), (&(POINTS[i].col)));

  rc_maxmin_ret (&mncol, &mxcol, &mnrow, &mxrow);

/**********************************************/
/* Initialize the variables the bitmap needs  */
/* and create a cleared bitmap with enough     */
/* memory allocated to hold all segments in    */
/* in the given zone and a two row/col buffer  */
/* of clear bits.                              */
/**********************************************/
  bitmap_init (&mnrow, &mncol, &mxrow, &mxcol);
  bitmap = create_bitmap ();

/**********************************************/
/* For each segment in the array of records,  */
```

F-23

```
      /* determine the bit row and bit index into    */
      /* the bitmap and set that bit.                 */
      /***********************************************/
        for (i=0;i<count;i++)
        {
          convert_rc_to_bm ((&(POINTS[i].row)), (&(POINTS[i].col)),
                             &bitrow, &bitcol);

          set_bit (bitmap, &bitrow, &bitcol);
        }

      /***********************************************/
      /* Now that the bitmap has been created and    */
      /* filled, open a bitmap file and write the    */
      /* bitmap to disk. Close the bitmap file when  */
      /* finished.                                   */
      /***********************************************/
        filedesc = open_bitmap_file (memory_filename);
        write_out_bitmap (&filedesc, bitmap);
        close_bitmap_file (&filedesc);

      /***********************************************/
      /* Now that the bitmap has been saved, return  */
      /* bitmap's memory back to the system.         */
      /***********************************************/
        destroy_bitmap     (bitmap);

      /***********************************************/
      /* Read in the bitmap from the memory file.    */
      /***********************************************/
        filedesc = open_bitmap_file (memory_filename);
        bitmap   = read_in_bitmap (&filedesc);
        close_bitmap_file (&filedesc);

      /*************************************************/
      /* Dump the bitmap to the ASCII file and close it */
      /*************************************************/
        dump_bitmap (fp, bitmap);
        fclose (fp);          /* ASCII Bitmap Dump File */

        return;
    }
```

2 BITMAP_SEGMENTSDAT.C
 Contains a suite of C language routines that build bitmaps from
 the SEGMENTS.DAT file. The SEGMENTS.DAT file is defined for the
 current processing thread. Subtopics include requirements and
 individual routines within the suite.

3 Include_Files
 The following file must be included for use with

BITMAP_SEGMENTSDAT.C:

V2_DIR:DATA_DEFS.H   Contains type definition for common
                        variables (e.g., scale, zone)

The following subroutines are required for linkage. For
additional information about each subroutine, see MDFFHELP under
the main topic CAC_SOURCE_CODE.

     BITMAP_APPL.C   A suite of high-level routines that perform
                    a variety of complex bitmap operations.

     BITMAP_PROCS.C   A suite of low-level routines that perform a
                    variety of basic operations on a bitmap.

     LL_MAXMIN.C   A suite of routines that determine the true
                    pair of minimum and maximum coordinates.

     QUAD_PROCS.C   A suite of routines that compute the minimum
                    and maximum latitude/longitude coordinates for
                    polar data.

     RC_MAXMIN.C   A suite of routines that determine the true
                    pair of minimum and maximum row/column values
                    from a set of row/column pairs.

3 Suite_Description
BITMAP_SEGMENTSDAT.C contains the following subroutines:

4 BUILD_SEGMENTSDAT_BITMAPS
Converts each zone within the SEGMENTS.DAT file to a bitmap and
places pointers to these bitmaps in an array. The array is
dimensioned from 0 to (number of zones - 1). If a zone is not
present within the SEGMENTS.DAT file, the array position for that
zone will be NULL. The following syntax is used to invoke
build_segmentsdat_bitmaps:

```
void build_segmentsdat_bitmaps (SCALE *mapscale,
                                BIT *coverage[NUM_ZONES])
```

where
                BIT   The Bitmap for the specified scale and zone
                        (returned)
           mapscale   Contains the chart scale

coverage[NUM_ZONES]   Contains the bitmap for the area of
                         coverage
                         (returned)

5 Example
The following code segment provides an example of usage.

```
     BIT    maps[NUM_ZONES];
  SCALE     scale;

   scale = 4;

   build_segmentsdat_bitmaps (&scale, maps);

  /* If the SEGMENTS.DAT file contains two zones, for example */
  /* 2 and 3, the array (maps) will look like this: */

   maps[0] = NULL
   maps[1] = NULL
   maps[2] = (some address to a bitmap containing zone 2 data)
   maps[3] = (some address to a bitmap containing zone 3 data)
   maps[4] = NULL
```

4 COVERAGE_FROM_SEGMENTSDAT

Returns a bitmap of the data in the SEGMENTS.DAT file for a
specified scale and zone. The minimum/maximum row/column values
of the bitmap are returned. The following syntax is used to
invoke coverage_from_segmentsdat:

```
   BIT *coverage_from_segmentsdat (SCALE *mscale, ZONE *zone,
                                   COL *mncol, COL *mxcol,
                                   ROW *mnrow, ROW *mxrow)
```

Where

BIT	The Bitmap for the specified scale and zone (returned)
mscale	Contains the chart scale (passed)
zone	Contains a TS zone (passed)
mncol,mxcol	Contains minimum and maximum column values (returned)
mnrow,mxrow	Contains minimum and maximum row values (returned)

5 Example

The following code segment provides an example of usage.

```
     BIT    *map;
  SCALE     scale;
   ZONE     zone;

 scale = 4;
 zone  = 3;

  /* This invocation will return a bitmap of the data in the     */
  /* SEGMENTS.DAT file for a specified scale and zone.  The       */
  /* minimum/maximum row/column values of the bitmap are returned.*/
```

```
map = coverage_from_segmentsdat (&scale, &zone, &mincol, &maxcol,
                                 &minrow, &maxrow);
```

2 BITMAP_SOURCE.C
Contains a suite of C language routines that build bitmaps from
downsampled and compressed segment data. Subtopics include
requirements and individual routines within the suite.

3 Include_Files
The following files must be included for use with BITMAP_SOURCE.C:

V2_DIR:BITMAP_SOURCE.H   Contains data definitions used by the
                        subroutines.

V2_DIR:DATA_DEFS.H   Contains type definitions for common
                        variables (e.g., scale, zone)

The following routines are required for linkage. For additional
information about each subroutine, see MDFFHELP main topic
CAC_SOURCECODE.

BITMAP_APPL.C   A suite of high-level routines that perform
                 a variety of complex bitmap operations.

BITMAP_PROCS.C   A suite of low-level routines that perform a
                 variety of basic operations on a bitmap.

LL_MAXMIN.C   A suite of routines that determine the true
                 pair of minimum and maximum coordinates.

QUAD_PROCS.C   A suite of routines that compute the minimum
                 and maximum latitude/longitude coordinates for
                 polar data.

RC_MAXMIN.C   A suite of routines that determine the true
                 pair of minimum and maximum row/column values
                 from a set of row/column pairs.

3 Suite_Description
BITMAP_SOURCE.C contains the following subroutines:

4 BITMAP_DS_SOURCE
Is passed a device name that contains downsampled data (i.e.
CHART_SEGMENTS:) and builds a bitmap of this data. The bitmap,
built with the minimum and maximum row/column values that were
also passed as arguments, is returned. The following syntax is
used to invoke bitmap_ds_source:

```
BIT *bitmap_ds_source (char *s,    SCALE *mscale, ZONE *zone,
                            COL *mncol, COL *mxcol,
                            ROW *mnrow, ROW *mxrow)
```
where

BIT	The returned bitmap
s	Contains the device name (passed)
mscale	Contains the map scale (passed)
zone	Contains the TS zone number (passed)
mncol	Contains the minimum column number (passed)
mxcol	Contains the maximum column number (passed)
mnrow	Contains the minimum row number (passed)
mxrow	Contains the maximum row number (passed)

4 BITMAP_MAP_SOURCE

Is passed a device name (i.e., CHART_ODI_DISK: or CDROM01:), and
builds a bitmap of the data that are contained on this device.
The
bitmap, having the minimum and maximum row/column values that were
passed as arguments, is returned. The following syntax is used to
invoke bitmap_map_source:

```
BIT *bitmap_map_source (char *device, SCALE *mscale,  ZONE *zone,
                            COL *mncol,       COL *mxcol,
                            ROW *mnrow,       ROW *mxrow)
```
where

BIT	The returned bitmap
device	Contains the device name (passed)
mscale	Contains the map scale (passed)
zone	Contains the TS zone number (passed)
mncol	Contains the minimum column number (passed)
mxcol	Contains the maximum column number (passed)
mnrow	Contains the minimum row number (passed)
mxrow	Contains the maximum row number (passed)

5 Example
The following code segment provides an example of usage.

```
/* path to data to be placed in bitmap. */
```

```
     path = "CDROM01:";

    /* call this first....located in CAC_UTIL.C. */
     cac_init (path);

    /* Get scale and zone. */
     scale = cac.map_scale;

    /* Get the minimum and maximum row/col values of the data. */
     get_minmax_from_cacid (&zone, &mncol, &mxcol, &mnrow, &mxrow);

    /* Return (i.e., create) a bitmap that is named map. */
     map  = bitmap_map_source (path,    &scale, &zone,
                                    &mncol, &mxcol,
                                    &mnrow, &mxrow);
```

4 GET_MINMAX_FROM_CACID
Extracts the minimum and maximum row/col values from a CAC CD-ROM.
The function "cac_init" (see subtopic CAC_UTIL.C) must be invoked
prior to use of this function. The following syntax is used to
invoke get_minmax_from_cacid:

```
    void get_minmax_from_cacid (short *zone,
                                   COL *mncol, COL *mxcol,
                                   ROW *mnrow, ROW *mxrow)
```

  where
          zone   Contains the TS zone number
                  (passed)
          mncol   Contains the minimum column number
                  (returned)
          mxcol   Contains the maximum column number
                  (returned)
          mnrow   Contains the minimum row number
                  (returned)
          mxrow   Contains the maximum row number
                  (returned)

5 Example
 The following code segment provides an example of usage.

```
    /* path to data to be placed in bitmap. */
     path = "CDROM01:";

    /* call this first....located in CAC_UTIL.C. */
     cac_init (path);

    /* Get scale and zone. */
     scale = cac.map_scale;

    /* Get the minimum and maximum row/col values of the data. */
     get_minmax_from_cacid (&zone, &mncol, &mxcol, &mnrow, &mxrow);
```

```
/* Return (i.e., create) a bitmap that is named map. */
 map   = bitmap_map_source (path,    &scale, &zone,
                                     &mncol, &mxcol,
                                     &mnrow, &mxrow);
```

4 GET_MINMAX_FROM_DS
 Is passed a device name where downsample segments are housed
 (e.g., CHART_SEGMENTS:) and returns the minimum and maximum
 row/col values of the data. The following syntax is used to
 invoke get_minmax_from_ds:

```
void get_minmax_from_ds (char *device, SCALE *mscale, ZONE *zone,
                         COL *mncol,      COL *mxcol,
                         ROW *mnrow,      ROW *mxrow)
```
 where
 >   device   Contains the device name
 >   (passed)
 >   mscale   Contains the map scale
 >   (passed)
 >   zone   Contains the TS zone number
 >   (passed)
 >   mncol   Contains the minimum column number
 >   (returned)
 >   mxcol   Contains the maximum column number
 >   (returned)
 >   mnrow   Contains the minimum row number
 >   (returned)
 >   mxrow   Contains the maximum row number
 >   (returned)

4 GET_MINMAX_FROM_MAP
 Is passed a device name (i.e., CHART_ODI_DISK: or CDROM01:), and
 returns the minimum and maximum row/col values of the data that
 are contained in the MAP Directory. The following syntax is used
 to invoke get_minmax_from_map:

```
void get_minmax_from_map (char *device, SCALE *mscale, ZONE *zone,
                          COL *mncol,      COL *mxcol,
                          ROW *mnrow,      ROW *mxrow)
```
 where
 >   device   Contains the device name
 >   (passed)
 >   mscale   Contains the map scale
 >   (passed)
 >   zone   Contains the TS zone number
 >   (passed)
 >   mncol   Contains the minimum column number
 >   (returned)
 >   mxcol   Contains the maximum column number
 >   (returned)
 >   mnrow   Contains the minimum row number

F-30
```

                    (returned)
          mxrow    Contains the maximum row number
                    (returned)


2 BITMAP_TRANS.C
 Combines two or more differently sized bitmaps into one bitmap.
 The bitmaps that are to be combined must be stored on disk.

 BITMAP_TRANS.C contains three subroutines.  In order to work
 correctly, all three routines must be called in proper order.

3 Include_Files
 The following files must be included for use with BITMAP_TRANS.C:

  MDFFSCR:BITMAP_TRANS.H  Contains data definitions used by the
                          subroutines.
  MDFFSCR:DATA_DEFS.H     Contains type definition for common
                          variables (e.g., scale, zone)


 The following subroutines are required for linkage.  For
 additional information about each subroutine, see MDFFHELP under
 the main topic CAC_SOURCE_CODE.

   BITMAP_APPL.C   A suite of high-level routines that perform a
                   variety of complex bitmap operations.

   RC_MAXMIN.C     A suite of routines that determine the true pair
                   of minimum and maximum row/column values from a
                   set of row/column pairs.

   BITMAP_PROCS.C  A suite of low-level routines that perform a
                   variety of basic operations on a bitmap.


3 Example
 The following pseudocode provides an example correct usage.

 call BITMAP_TRANS_INIT (scale, zone) ! This sets up the variables
                                      ! used by the next two calls.
                                      ! This routine MUST be called
                                      ! first. Scale and zone are
                                      ! INTEGER*4 and passed by
                                      ! reference.

 ! Loop through the number of bitmaps that are to be combined into
 ! one bitmap

 LOOP 1 TO Number_Of_Bitmaps


F-31

```
        ! Send in as many bitmap filenames as you like.
        ! The filename must include the complete path.
        ! Filename is a character string pointer and is
        ! passed by reference.

        call BITMAP_TRANS_SEND (filename)

   END LOOP

   ! This is the final routine and MUST be called last.  It returns
   ! the one combined bitmap and its minimum/maximum row/column
   ! values.  Arguments are passed by reference

    bitmap = bitmap_trans_ret (mincol, maxcol, minrow, maxrow)
```

3 Suite_Description
BITMAP_TRANS.C contains the following subroutines:

4 BITMAP_TRANS_INIT
This routine must be invoked first. It is used to initialize
variables used by the other routines. The following syntax is used
to invoke BITMAP_TRANS_INIT:

```
   void bitmap_trans_init (SCALE *mapscale, ZONE *mapzone)
```

where

mapscale	INTEGER*4	Contains the chart (i.e., map) scale
		(passed by reference)
mapzone	INTEGER*4	Contains a TS zone
		(passed by reference)

4 BITMAP_TRANS_SEND
This routine is used to specify filenames of the bitmaps that are
to be combined. Hence, it should be invoked iteratively; each of
these bitmaps will be combined to form one larger bitmap. The
bitmap filenames must include the complete path specification
(i.e., device and directory names). The following syntax is used
to invoke BITMAP_TRANS_SEND:

```
   void bitmap_trans_send (char *filename)
```

where

filename	contains a character string pointer
	(passed by reference)

4 BITMAP_TRANS_RET
This is the final routine and MUST be called last. It returns the
one combined bitmap and its minimum and maximum row/column values.

The text "UNCLASSIFIED" appears vertically in the top right margin.

The following syntax is used to invoke BITMAP_TRANS_RET:

```
BIT *bitmap_trans_ret (COL *mncol, COL *mxcol,
                       ROW *mnrow, ROW *mxrow)
```
where
mncol,mxcol   Contains the minimum and maximum column values (returned)

mnrow, mxrow   Contains the minimum and maximum row values (returned)

2 BUILD_CD_ID.C

---------- TO BE ADDED LATER ------------

2 CAC.C

CAC.C contains a suite of C language routines that provide portable support for reading CAC CD-ROMs. Individual routines include:

CAC_INIT: Initializes the CAC retrieval software.

CAC_INQ_PALETTE: Returns the "day" or "night" color palette for current palette id.

CAC_GET_LL: Returns the color of the pixel at the specified latitude and longitude. Also returns the address of the entire decompressed segment, if desired.

CAC_GET_RC: Returns the address of the entire decompressed segment specified by the row, column, and map zone.

BUFFER_COMPRESSED_SEGMENT: Low level routine used by "CAC_GET_LL" and "CAC_GET_RC". This routine SHOULD NOT be called by users at a high level! See MAIN_LL.C and MAIN_RC.C for examples on implementation of CAC reader software.

Subtopics provide more detailed descriptions of each routine within the suite.

3 CAC_INIT

Reads the [ID]CD_ID.DAT and [ID]CD_COVRG.DAT files from the specified device. Initializes the internal CAC work structure based on the contents of the [ID]CD_ID.DAT and [ID]CD_COVRG.DAT files.

4 Invocation

CAC_INIT is invoked with the following syntax:

```
short CAC_INIT (char cac_device[])
```

Where
    cac_device: Name of device that CAC CD-ROM is loaded on.
               (passed, char[])

CAC_INIT returns one of the following status values:

Status	Meaning
1	Normal
-1	Error reading [ID]CD_ID.DAT
-2	Error reading [ID]CD_COVRG.DAT
-3	CDROM is NOT a CAC CDROM

3 CAC_INQ_PALETTE

Returns the color palette for the specified PA ID. Reads the
color palette file for the specified palette ID. The color
palette size is returned along with three arrays containing
the red, green, and blue components of the specified color
palette. In addition, the "day" or "night" color palette may be
returned.

4 Invocation

CAC_INQ_PALETTE is invoked with the following syntax:

```
short CAC_INQ_PALETTE (char type,
                       short palid,
                       short *size,
                       byte red[],
                       byte green[],
                       byte blue[])
```

Where
      type: Type of palette to load (DAY or NIGHT)
           (passed, char)

     palid: Palette identification. This is a four digit number
           identifying the color palette to use for the
           selected segment.
           (passed, int)

      size: Size of the color palette returned.
           (returned, short *)

       red: Array of size "size" containing the RED component of
           the color palette.
           (returned, char [])

    green: Array of size "size" containing the RED component of
           the color palette.
           (returned, char [])

blue: Array of size "size" containing the RED component of
      the color palette.
      (returned, char [])

CAC_INQ_PALETTE returns one of the following status values:
        Status: Meaning
            1 : Normal
           -1 : Error opening PALETTE.DAT file
           -2 : Error reading PALETTE.DAT file

3 CAC_GET_LL

Returns the value of the pixel at the specified latitude/longitude
position.   Also supplies a pointer to the entire decompressed
segment if requested.

4 Invocation

CAC_GET_LL is invoked with the following syntax:

        short CAC_GET_LL (float lon,
                          float lat,
                          short *palid,
                          short *color)

  Where:
        lon: Longitude of requested pixel.
             (passed, float)

        lat: Latitude of requested pixel.
             (passed, float)

        palid: Palette identification of pixel at lat/lon.
               (returned, short *)

        color: Pixel value at specified lat/lon.   This is the index
               into the color palette.
               (returned, short *)

  CAC_GET_LL returns one of the following status values:
        Status: Meaning
            1: Normal
           -1: Specified lat/lon does NOT fall with bounds
               specified in the CD_COVRG.DAT file. The data is
               NOT on this CD-ROM.

3 CAC_GET_RC

Returns the decompressed segment at the specified row/column. The
map zone is required in order to process zone overlap correctly.

4 Invocation

F-35

CAC_GET_RC is invoked with the following syntax:

```
short CAC_GET_RC (long row,
                  long col,
                  short  map_zone,
                  short  *palid)
```

Where

row: Row of requested segment.
(passed, long)

col: Column of requested segment.
(passed, long)

map_zone: TS map zone that the requested segment is in.
This is used to allow specifying segments in
zone overlap areas.
(passed, short)

palid: Palette identification of segment at row/column.
(returned, short *)

CAC_GET_RC returns one of the following status values:
Status: Meaning
1: Normal
-1: Specified map zone is NOT on this CDROM.
-2: Specified segment at row/column is NOT on this CD-ROM.

3 BUFFER_COMPRESSED_SEGMENT
Performs the reading and decompression of a segment.
Low level routine used by "cac_get_ll" and "cac_get_rc" to
decompress a segment. This routine is NOT user callable.

4 Invocation
BUFFER_COMPRESSED_SEGMENT is invoked with the following syntax:

```
short BUFFER_COMPRESSED_SEGMENT()
```

There are NO arguments. All information to read and decompress
a segment is in the structure "CAC".

BUFFER_COMPRESSED_SEGMENT returns one of the following status
values:
Status: Meaning
1: Normal

2 CAC_MISC.C
Contains low-level CAC reader software that is written in C
programming language. Most users should not require these
routines: the high-level CAC reader routines (see subtopic

CAC_UTIL) will be sufficient for most applications.

3 Include_files
The following files must be included in source code.

#include <stdio.h>	Standard C I/O library
#include "ansi.h"	Enables software portability to the VAX/VMS platform
#include "sysdep.h"	Enables software portability to the VAX/VMS platform
#include "sysutil.h"	Enables software portability to the VAX/VMS platform
#include "m4_const.h"	TS-specific definitions
#include "cac.h"	CAC-specific definitions
#include "areasorc.h"	Structure for data in the [ID]AREASORC.DAT file
#include "areadrc.h"	Structure for data in the [ID]AREADRC.DAT file
#include "cd_header.h"	Structure for data in the [MAPx.CDxxxxx]HEADER.DAT file
#include "dr_header.h"	Structure for data in the [MAPx.CDxxxxx.TPCxxxxx]HEADER.DAT file
#include "sg_header.h"	Structure for data in the [MAPx.CDxxxxx.TPCxxxxx]SGHED.DATfile
#include "pa.h"	Palette area and data from CD_COVRG.DAT file

3 READ_AREADRC
Reads the contents of the specified AREADRC.DAT file. The
contents are read into the structure pointed to by "areadrc".

4 Invocation
Read_areadrc is invoked using the following syntax:

short read_areadrc (char path[], structr areadrc *areadrc,
                    short *numpas)
where
     path: Complete file specification of the AREADRC.DAT file.
           (char [], passed)

areadrc: Structure to contain data read from AREADRC.DAT file.
                (struct areadrc *, returned)

        numpas: Number of PA areas (zones) in the AREADRC.DAT file.
                (short *, returned)

4 Returns
 Read_areadrc returns a file access status containing one of the
 following values:
        Value   Meaning
          1   Normal termination
         -1   Error opening AREADRC.DAT file.
         -2   Error reading AREADRC.DAT file.
        (returned, short)

3 READ_AREASORC
 Reads the contents of the specified AREASORC.DAT file. The
 contents are read into the structure pointed to by "areasorc".

4 Invocation
 Read_areasorc is invoked using the following syntax:

 short read_areasorc (char path[], structr areasorc *areasorc,
                        short *numpas)
 where
        path: Complete file specification of the AREASORC.DAT file.
                (char [], passed)

      areasorc: Structure to contain data read from AREASORC.DAT file.
                (struct areasorc *, returned)

        numpas: Number of PA areas (zones) in the AREASORC.DAT file.
                (short *, returned)
4 Returns
 Read_areasorc returns a file access status containing one of the
 following values:
        Value   Meaning
          1   Normal termination
         -1   Error opening AREASORC.DAT file.
         -2   Error reading AREASORC.DAT file.
        (returned, short)

3 READ_CDHEADER
 Reads the contents of the specified CDHEADER.DAT file. The
 contents are read into the structure pointed to by "cdheader".

4 Invocation
 Read_cdheader is invoked using the following syntax:

 short read_cdheader (char path[], structr cdheader *cdheader,
                        short *numpas)
 where

path: Complete file specification of the CDHEADER.DAT file.
(char [], passed)

cdheader: Structure to contain data read from CDHEADER.DAT file.
(struct cdheader *, returned)

4 Returns
Read_cdheader returns a file access status containing one of the
following values:

Value	Meaning
1	Normal
-1	Error opening CDHEADER.DAT file.
-2	Error reading CDHEADER.DAT file.

(returned, short)

3 READ_DRHEADER
Reads the contents of the specified DRHEADER.DAT file. The
contents are read into the structure pointed to by "drheader".

4 Invocation
Read_drheader is invoked using the following syntax:

```
short read_drheader (char path[], structr drheader *drheader,
                     short *numpas)
```
where
path: Complete file specification of the DRHEADER.DAT file.
(char [], passed)

drheader: Structure to contain data read from DRHEADER.DAT file.
(struct drheader *, returned)

4 Returns
Read_drheader returns a file access status containing one of the
following values:

Value	Meaning
1	Normal termination
-1	Error opening DRHEADER.DAT file.
-2	Error reading DRHEADER.DAT file.

(returned, short)

3 READ_SGHEADER
Reads the contents of the specified SGHEADER.DAT file. The
contents are read into the structure pointed to by "sgheader".

4 Invocation
Read_sgheader is invoked using the following syntax:

```
short read_sgheader (char path[], structr sgheader *sgheader,
                     short *numpas)
```
where
path: Complete file specification of the SGHEADER.DAT file.
(char [], passed)

F-39

sgheader: Structure to contain data read from SGHEADER.DAT file.
(struct sgheader *, returned)

4 Returns
Read_sgheader returns a file access status containing one of the
following values:
    Value   Meaning
       1   Normal termination
      -1   Error opening SGHEADER.DAT file.
      -2   Error reading SGHEADER.DAT file.
    (returned, short)

3 FILE_OPEN_ERROR
Prints a file open error message and returns an error status
(whose value is -1). File_open_error is invoked using the
following syntax:
        static short file_open_error (char path[])
where
        char message[80] The error message

3 FILE_READ_ERROR
Prints a file read error message and returns an error status
(whose value is -2). File_read_error is invoked using the
following syntax:
        static short file_read_error(char path[])
where
        char message[80] The error message

2 CAC_UTIL.C
A suite of high-level C language utilities for reading CAC
CD-ROMs. See Overview for brief descriptions of each routine.

See subtopics MAIN_LL.C and MAIN_RC.C for implementation examples
of CAC reader software.

3 Overview
The suite of C language utilities, with brief descriptions,
include:

                DECODE_KEY: Converts "key" name to row/column values.

        DECOMPRESS_SEGMENT: Reads and decompress the specified
                            segment.

             DOUBLE_TO_SI: Converts a double precision number to
                           a scaled integer number.

               ENCODE_KEY: Converts row/column values to "key" name.

                   EQ2POL: Rotates equatorial latitude and longitude

to the polar zone.

GET_DECOMPRESSED_PIXEL: Retrieves specified pixel from a compressed segment.

GET_SEGMENT_NAME: Builds the CAC compressed segment path name.

INIT_MEM: Initializes memory required to buffer compressed segment data.

LATLON_CALC: Convert row and column values to latitude and longitude.

LOAD_LEGEND_DATA: Reads header, palette, and image files for specified legend data.

ODD: Determines if a number is even or odd.

POL2EQ: Rotates polar latitude and longitude to the equatorial zone.

RC_CALC: Converts latitude and longitude to row and column values.

READ_CD_COVRG: Reads the contents of the specified CD_COVRG.DAT file.

READ_CD_ID: Reads the CD_ID.DAT file from the CD-ROM.

READ_COMPRESSED_SEGMENT: Reads the compressed segment and its codebook.

READ_PA_COVERAGE: Reads the PA coverage file (COVERAGE.DAT).

READ_PALETTE: Read the specified color palette.

SI_CONVERT: Converts an ASCII string to a scaled integer number.

SI_TO_DOUBLE: Converts scaled integer numbers to double precision numbers.

SPDEC: Decompresses CAC data.

3 DECODE_KEY
Decodes the key name for downsampled and compressed segments into its row and column components.

4 Invocation
DECODE_KEY is invoked using the following syntax:

```
            void decode_key (key, row, col)
```

Where

        key: Downsampled segment key name
             (passed unsigned long)

        row: Downsampled segment row number
             (returned long)

        col: Downsampled segment column number
             (returned long)

4 Returns

DECODE_KEY returns the downsampled segments row and column numbers through the following arguments:

        row: Downsampled segment row number,
             (returned long)

        col: Downsampled segment column number
             (returned long)

3 DECOMPRESS_SEGMENT

Reads a compressed segment and its codebook and then decompresses the segment. A pointer to the beginning of the decompressed segment is returned.

4 Invocation

DECOMPRESS_SEGMENT is invoked using the following syntax:

```
short decompress_segment (char pa_path[], unsigned char
                              *decomp_seg)
```

Where

    File access status is returned through a short variable.

        pa_path: Complete file specification of CAC segment
              file to decompress.
              (passed char [])

      decomp_seg: Pointer to the beginning of the array containing
              the decompressed segment data.
              (returned unsigned char)

4 Returns

DECOMPRESS_SEGMENT returns

        File access status containing one of the following values:

Value	Meaning
1	Normal
-1	Error opening compressed CAC segment file.

```
              -2      Error reading compressed CAC segment codebook.
              -3      Error reading compressed CAC segment data.
          (returned short)

      decomp_seg: Pointer to the beginning of the array containing
                  the decompressed segment data.
                  (returned unsigned char)
```

3 DOUBLE_TO_SI
Converts a double precision number to a 32-bit scaled integer
number. Hence, this routine reduces the precision of the data.

4 Invocation
DOUBLE_TO_SI is invoked using the following syntax:

```
    double double_to_si (double value)
```

Where:
> The scaled integer is returned through a 32-bit
> (double) variable.

> value: The double precision value to be converted.
>   (passed double)

3 ENCODE_KEY
Encodes the key name for downsampled and compressed segment files.

4 Invocation
ENCODE_KEY is invoked using the following syntax:

```
    unsigned long encode_key (row, col)
```

Where
> The key name is returned through an unsigned long
> variable.

> row: Segment row number
>   (passed long)

> col: Segment column number
>   (passed long)

3 EQ2POL
Rotates equatorial latitude and longitude coordinates into
polar zone coordinates.

4 Invocation
EQ2POL is invoked using the following syntax:

```
    void eq2pol (double *dlatin,  double *dlngin,
```

```
              double *dlatout, double *dlngout,
              short *zone)
    Where
        dlatin: Segments equatorial latitude coordinate.
                (passed double)

        dlngin: Segment's equatorial longitude coordinate.
                (passed double)

       dlatout: Segment's rotated polar latitude coordinate.
                (returned double)

       dlngout: Segment's rotated polar longitude coordinate.
                (returned double)

          zone: The polar zone number.
                (passed short)
```

4 Returns
 EQ2POL returns the rotated coordinate values through the following
 arguments:

```
       dlatout: Segment's rotated polar latitude coordinate.
                (returned double)
       dlngout: Segment's rotated polar longitude coordinate.
                (returned double)
```

3 GET_DECOMPRESSED_PIXEL
 Retrieves specified pixel from a compressed segment.

4 Invocation
 GET_DECOMPRESSED_PIXEL is invoked using the following syntax:

```
       short get_decompressed_pixel(short y, short x)
```

   Where
          The pixel value is returned through a short variable.

          y: Y Coordinate in the decompressed segment
             (passed short)

          x: X coordinate in the decompressed segment
             (passed short)

4 Returns
   GET_DECOMPRESSED_PIXEL returns the specified pixel from
   the decompressed segment as a short variable.

3 GET_SEGMENT_NAME
 Builds the CAC compressed segment path name from the palette area

                              F-44

directory name, row number, column number, and zone of the
requested segment.

4 Invocation
  GET_SEGMENT_NAME is invoked using the following syntax:

     void get_segment_name (char pa_path[], long row, long col,
                            short zone, char seg_path[])

   Where
     pa_path: Path to the palette area subdirectory. Note the
              "period" at the end of the VMS path name.
              (i.e., for   VMS: CDROM:[MAP3.PA012901.)
              (i.e., for UNIX: /cdrom/map3/pa012901/)
              (i.e., for MSDOS: D:\map3\pa012901\)
              (passed char [])

         row: Row number of segment to decompress.
              (passed long)

         col: Column number of segment to decompress.
              (passed long)

        zone: Tesselated Sphere zone number corresponding to
              "pa_path".
              (passed short)

    seg_path: Complete path specification for requested segment.
              (i.e., for   VMS:
                     CDROM:[MAP3.PA012901.R000015]12345678.214)
              (i.e., for UNIX:
                     /CDROM/MAP3/PA012901/R000015/12345678.214)
              (i.e., for MSDOS: D:\MAP3\PA012901\R000015\12345678.214)

              (returned char [])

4 Returns
  GET_SEGMENT_NAME returns the complete path specification,
  "seg_path", for the requested segment.

    seg_path: Complete path specification for requested segment.
              (ie. for VMS:
                     CDROM:[MAP3.PA012901.R000015]12345678.214)
              (ie. for UNIX:
                     /CDROM/MAP3/PA012901/R000015/12345678.214)
              (ie. for MSDOS:
                     D:\MAP3\PA012901\R000015\12345678.214)

              (returned char [])

3 INIT_MEM

Initializes the memory required to buffer compressed segment data.
This routine should only be executed once.

The number of CAC compressed segments that may be buffered at one
time is controlled by #define MAX_SEGS_IN_BUF (which is defined in
the file CAC.H). The structure element "mem_init", within the
structure "cac" (also defined in CAC.H), is set when this routine
terminates.

4 Invocation
INIT_MEM is invoked using the following syntax:

    init_mem()

 Note that there are no arguments or return values.

3 LATLON_CALC
 Calculates latitude and longitude coordinates from row and column
 values. The zone is required for handling overlap areas.

4 Invocation
LATLON_CALC is invoked using the following syntax:

   void latlon_calc (short *zone, short *scale, long *row,
                     long *col, double *rlatp, double *rlonp)

  Where
          zone : TS zone number to be used in the conversion.
                 (passed short integer)

          scale: Map scale.
                 (passed short integer)

            row: TS segment row number to convert.
                 (passed long)

            col: TS segment column number to convert.
                 (passed long)

          rlatp: Latitude coordinate based on zone, scale row, col
                 (returned double)

          rlonp: Longitude coordinate based on zone, scale row, col
                 (returned double)

4 Returns
 LATLON_CALC returns the segment's latitude and longitude
 coordinates through the following arguments:

          rlatp: Segment's latitude coordinate
                 (returned double)

F-46

rlonp: Segment's longitude coordinate
(returned double)

3 LOAD_LEGEND_DATA

Reads header, palette, and image files for the specified legend
data. A pointer to the beginning of the legend image is returned
along with the RGB buffers and the size (in rows/columns) of the
legend image itself.

4 Invocation

LOAD_LEGEND_DATA is invoked using the following syntax:

```
short load_legend_data (char legend_path[], byte *legend_buf,
                        byte rbuf[], byte gbuf[], byte bbuf[],
                        unsigned long *legend_x, unsigned long
                        *legend_y)
```

Where

File access status is returned through a short variable.

legend_path: File specification of the directory containing
the legend data.
(passed char [])

legend_ptr: Pointer to the beginning of the array
containing the legend image data.
(returned byte)

rbuf: Red component of the legend image's palette.
(returned byte [])

gbuf: Blue component of the legend image's palette.
(returned byte [])

bbuf: Green component of the legend image's palette.
(returned byte [])

legend_x: Size of the legend image in the "x" direction
(columns).
(returned unsigned long)

legend_y: Size of the legend image in the "y" direction
(rows).
(returned unsigned long)

4 Returns

LOAD_LEGEND_DATA returns the following information:

File access status containing one of the following values:
Value   Meaning
1   Normal
-1   Error opening legend header file.

F-47

-2   Error reading legend header file.
                        -3   Error opening legend image file.
                        -4   Error reading legend image file.
                  (returned short)

             legend_ptr: Pointer to the beginning of the array
                         containing the legend image data.
                         (returned byte)

                 rbuf: Red component of the legend image's palette.
                       (returned byte [])

                 gbuf: Blue component of the legend image's palette.
                       (returned byte [])

                 bbuf: Green component of the legend image's palette.
                       (returned byte [])

             legend_x: Size of the legend image in the "x" direction
                       (columns).
                       (returned unsigned long)

             legend_y: Size of the legend image in the "y" direction
                       (rows).
                       (returned unsigned long)

3 ODD
Determines whether or not the specified value is odd or even.

4 Invocation
Odd is invoked using the following syntax:

      short odd (short num)

   Where
         A short variable is used to return the status of "num"

         num: number to test
              (passed short)

4 Returns
   ODD returns a short variable that contains one of the following
   values:

         0 if the number tested is odd
         1 if the number tested is even

3 POL2EQ
Rotates polar latitude and longitude coordinates to equatorial
zone coordinates.

4 Invocation

POL2EQ is invoked using the following syntax:

```
void pol2eq (double *dlatin, double *dlngin, double *dlatout,
             double *dlngout)
```

Where
      dlatin: Segments polar latitude coordinate.
            (passed double)

      dlngin: Segment's polar longitude coordinate.
            (passed double)

    dlatout: Segment's rotated equatorial latitude coordinate.
            (returned double)

    dlngout: Segment's rotated equatorial longitude coordinate.
            (returned double)

4 Returns
POL2EQ returns the rotated coordinate values through the following arguments:

    dlatout: Segment's rotated equatorial latitude coordinate.
            (returned double)

    dlngout: Segment's rotated equatorial longitude coordinate.
            (returned double)

3 RC_CALC
Converts latitude and longitude to row and column values.

4 Invocation
```
void rc_calc (double *rlat, double *rlon, short *scale,
              short *zone, long *row, long *col)
```
  Where
        rlat: TS segment latitude coordinate
           (passed double)

        rlon: TS segment longitude coordinate
           (passed double)

      scale: Map scale
           (passed short)

       zone: TS zone number that the segment lies within.
           (passed short)

        row: TS segment row number
           (returned long)

        col: TS segment column number

(returned long)

4 Returns
RC_CALC returns the segments row and column numbers through the following arguments:

row: TS segment row number based on passed latitude, longitude, scale and zone
  (returned long)

col: TS segment column number based on passed latitude, longitude, scale and zone
  (returned long)

3 READ_CD_COVRG
Reads the contents of the specified CD_COVRG.DAT file, which contains the approximate coverages for each PA area on the CAC and the PA area's associated zone number.

Originally, zone numbers were not part of the CD_COVRG.DAT file and were added at a later date (beginning with CAC CD-1991-A-MAP3-1005). Hence, earlier versions of the CD_COVRG.DAT file do not contain zone numbers. The structure "no_zone_cacs", containing the PA area/zone number association for those CAC's, is used for accessing these "older" files. The CD_ID.DAT file MUST be read first in order to correctly process the CD_COVRG.DAT file (see CAC_INIT for information about file access).

4 Invocation
READ_CD_COVRG is invoked using the following syntax:

```
short read_cd_covrg (char path[],
                     char pa_nums[MAX_PAS][8],
                     short *num_pas,
                     double pa_latlon[MAX_PAS][4],
                     char pa_zones[MAX_PAS])
```
  Where
        File access status is returned through a short variable.

        path: Complete path specification to the CD-ROM's
              CD_COVRG.DAT file.
              (passed char [])

    pa_nums: Two-dimensional array of palette area names from the
             CD_COVRG.DAT file. Each palette area name is 8
             bytes. The maximum number of possible palette areas
             on one CD-ROM is MAX_PAS (see CAC.H).
             (returned char[][])

    num_pas: The number of palette areas on the CD-ROM.
             (returned short)

pa_latlon: Two-dimensional array of approximate coverages of each palette area on the CD-ROM. The order of the lat/lon data in the array is:
>   [*][0] - West longitude
>   [*][1] - East longitude
>   [*][2] - South latitude
>   [*][3] - North latitude
>   (returned double[][])

pa_zones: Array of TS zone numbers corresponding to the "pa_nums" above.
>   (returned char[][])

4 Returns
READ_CD_COVRG returns the following information:

File access status containing one of the following values:

Value	Meaning
1	Normal
-1	Error opening CD_COVRG.DAT file.
-2	Error reading the number of palette areas from CD_COVRG.DAT file.
-3	Error reading a palette area name from CD_COVRG.DAT file.
-4	Error reading a palette area lat/lon set from CD_COVRG.DAT file.

(returned short)

pa_nums: Two-dimensional array of palette area names from the CD_COVRG.DAT file. Each palette area name is 8 bytes. The maximum number of possible palette areas on one CD-ROM is MAX_PAS (see CAC.H).
>   (returned char[][])

num_pas: The number of palette areas on the CD-ROM.
>   (returned short)

pa_latlon: Two-dimensional array of approximate coverages of each palette area on the CD-ROM. The order of the latitude/longitude data in the array is:
>   [*][0] - West longitude
>   [*][1] - East longitude
>   [*][2] - South latitude
>   [*][3] - North latitude
>   (returned double[][])

pa_zones: Array of TS zone numbers corresponding to the "pa_nums" above.
>   (returned char[][])

3 READ_CD_ID
Reads the CD_ID.DAT file from the [ID] directory on the CD-ROM.

4 Invocation
READ_CD_ID is invoked using the following syntax.

    short read_cd_id (char path[], char data[])

Where
  Completion status is returned through a short variable.

    path: Complete path specification to the CD-ROM's CD_ID.DAT
          file.
          (passed char [])

    data: Contents of specified CD_ID.DAT file.
          (returned char [], requires twenty bytes)

4 Returns
READ_CD_ID returns the following information:

  File access status containing one of the following values:
      Value Meaning
        1   Normal
       -1   Error opening CD_ID.DAT file.
       -2   Error reading CD_ID.DAT file.
      (returned short)

    data: Contents of specified CD_ID.DAT file.
          (returned char [], requires twenty bytes)

3 READ_COMPRESSED_SEGMENT
  Reads the compressed segment and its codebook. The codebook and
  segment are buffered into the array "segbuf". This is an array of
  structures of type "ASegment". The number of segments that can be
  buffered is controlled by the definition of MAX_SEGS_IN_BUF (in
  CAC.H). This buffering reduces the overhead of having to re-read
  an often used segment.

4 Invocation
  READ_COMPRESSED_SEGMENT is invoked using the following syntax:

    short read_compressed_segment (char pa_path[],
                                   byte *codebook,
                                   byte *compseg)

  where
    File access status is returned through a short variable.

    pa_path: Complete file specification of CAC segment
             file of interest.
             (passed char [])

    codebook: Contains compression codebook data.
              (returned byte)

F-52

compseg: Contains compressed segment data.
(returned byte)

4 Returns:
READ_COMPRESSED_SEGMENT returns the following information:

File access status containing one of the following values:

Value Meaning
1 Normal
-1 Error opening compressed CAC segment file.
-2 Error reading compressed CAC segment codebook.
-3 Error reading compressed CAC segment data.
(returned short)

codebook: Contains compression codebook data.
(returned byte)

compseg: Contains compressed segment data.
(returned byte)

3 READ_PA_COVERAGE
Reads the scaled integer lat/long coordinate from a COVERAGE.DAT
file and returns them as doubles. Note that the scaled integers
are stored with their bytes swapped and must be re-swapped before
conversion to real numbers.

4 Invocation
READ_PA_COVERAGE is invoked using the following syntax:

short read_pa_coverage (pa_coverage_file, left_lon, right_lon,
bot_lat, top_lat)

Where
PA_COVERAGE_FILE: Path name specification of COVERAGE.DAT file.
DEVICE:[ODIXXXXX.MAPX.PAXXXXYY]COVERAGE.DAT
(passed char)

left_lon: Left longitude coordinate
(returned double)

right_lon: Right longitude coordinate
(returned double)

bot_lat: Bottom latitude coordinate
(returned double)

top_lat: Top latitude coordinate
(returned double)

4 Returns
READ_PA_COVERAGE returns the following information.

F-53

File access status is returned through a short variable
containing one of the following values:

    Value Meaning
      1 Normal
     -1 Error opening PA COVERAGE.DAT file.
     -2 Error reading PA COVERAGE.DAT file.
(returned short)

right_lon: Right longitude coordinate
        (returned double)

bot_lat: Bottom latitude coordinate
        (returned double)

top_lat: Top latitude coordinate
        (returned double)

3 READ_PALETTE
Reads the specified CAC day, night, or mono color palette.

4 Invocation
READ_PALETTE is invoked using the following syntax:

```
short read_palette (char type, char path[], byte red[],
                    byte green[], byte blue[])
```

Where
        file access status is returned through a short variable

type: Day or night palette differentiation
      (passed char)

path: Complete file specification of the CAC
      color palette file
      (passed char)

red: Red component of color palette
     (returned byte)

green: Green component of color palette
      (returned byte)

blue: Blue component of color palette
     (returned byte)

4 Returns
READ_PALETTE returns the following information:

File access status containing one of the following values.
    Value   Meaning
      1   Normal

```
        -1      Error opening CAC color palette
        -2      Error reading CAC color palette
        (returned short)
```

red: Red component of color palette
(returned byte)

green: Green component of color palette
(returned byte)

blue: Blue component of color palette
(returned byte)

3 SI_CONVERT

Converts an ASCII string in the form sDDDMMSS.SS to a scaled integer. Where

"s" is the sign of the latitude or longitude
(i.e., "+" or "-"). The "s" must always be present.

```
   DDD is degrees
    MM is minutes
 SS.SS is seconds
```

4 Invocation

SI_CONVERT is invoked using the following syntax:

```
long si_convert (*value,type)
```

Where:
The scaled integer of the "value" is returned through a long variable.

value: ASCII value of coordinates
(passed char)

type: Latitude or longitude type of conversion
(passed short)

3 SI_TO_DOUBLE

Converts a scaled integer to a real number (double). Returns the value as a double.

4 Invocation

SI_TO_DOUBLE is invoked using the following syntax:

```
double si_to_double (si)
```

Where:
The real number of "si" is returned through a double variable.

```
           si: Scaled integer
               (passed long)
```

3 SPDEC
Decompresses a CAC compressed segment.

4 Invocation

Due to the memory constraints of MS-DOS, this routine has two
syntaxes: One for MS-DOS and one for VMS and UNIX.

The VMS and UNIX syntax follows:

```
void spdec (unsigned char inptr[16384],
            unsigned char spcbptr[16384],
            unsigned char outptr[16384])
```

   Where
           inptr: Compressed segment to be decompressed. The
                  compressed segment is assumed to be 16382 bytes.
                  (passed byte)

         spcbptr: Codebook to be used to decompress the segment. The
                  codebook is assumed to be 1024 bytes.
                  (passed byte)

          outptr: Decompressed segment. The decompressed segment
                  requires 65536 bytes.
                  (returned byte)

2 CCOMMA.C
Places commas in an integer number. This function should only be
invoked by routines written in C code. For the equivalent FORTRAN
language function, see subtopic FCCOMMA.C

CCOMMA.C is invoked using the following syntax:

   long ccomma (number)

   where
           The function returns the address of a character string
           that contains the number (passed) with commas.

           number   Contains an integer number
                           (passed)

3 Example
The following pseudocode provides an example of usage.

   long   number=100100999   /* integer to be passed */

                              F-56
```

```
fprint ("%s /n") ccomma(number)      /* invocation */

/* The string "100,100,999" is printed */
```

2 CHECK_LON_ORIENTATION.C

A C language routine that determines whether a minimum longitude value and maximum longitude value are the true minimum and maximum values for a given data set.

3 Input_Output

In addition to minimum and maximum longitude values, CHECK_LON_ORIENTATION must be passed one of the following integer values:

> 0 - When the data set crosses 0 degrees longitude.

> 180 - When the data set crosses 180 degrees longitude.

CHECK_LON_ORIENTATION returns the following values:

> 1 (for TRUE) if the minimum longitude is LESS than the maximum longitude.

> 0 (or FALSE) if the minimum longitude is GREATER than the maximum longitude (when passed 0),
>
> OR
>
> if the minimum longitude is indeed GREATER than or EQUAL to the maximum longitude (when passed 180).

3 Invocation

CHECK_LON_ORIENTATION is invoked using the following syntax:

```
long CHECK_LON_ORIENTATION (long *status, float *minlon,
                            float *maxlon);
```

Where

> The orientation status is returned through a long (integer) variable.

> status: Whether the data set crosses 0 or 180
>   (passed long)
> minlon: Minimum longitude
>   (passed long)
> maxlon: Maximum longitude
>   (passed long)

Pseudocode examples of invocation from FORTRAN and C programs are provided.

4 FORTRAN_Example

FORTRAN pseudocode example:

F-57

```
integer*4    status = 180      ! Data set crosses 180 degrees
real*4       minlon = -50.0    ! Minimum longitude
real*4       maxlon =  60.0    ! Maximum longitude

if ((check_lon_orientation(status, minlon, maxlon)).ne.1) then
    temp   = minlon
    minlon = maxlon    ! minimum and maximum longitude values
    maxlon = temp      ! are not in the correct orientation
endif                  ! so they are swapped.
```

4 C_Example

C pseudocode example:

```
long check_lon_orientation (long *status, float *minlon,
                            float *maxlon);

status = 0;          /*  long, data set crosses 0 degrees */
minlon =  50.0;      /*  float, minimum longitude          */
maxlon = -60.0;      /*  float, maximum longitude          */

if (!(check_lon_orientation(&status, &minlon, &maxlon))
{
    temp = minlon;
    minlon = maxlon;  /* minimum and maximum longitude values*/
    maxlon = temp;    /* are not in the correct orientation  */
}                     /* so they are swapped.                */
```

2 CLEAN_UP.C

Subroutine CLEAN_UP.C traverses the current processing thread directory structure and performs functions necessary for trimming the chart ODI.

3 Functionality
CLEANUP.C performs the following functions that are necessary for trimming the chart ODI.

* Creates a log file in the MDFF_SCRATCH directory, named CLEANUP.LOG, which contains the following information:
    - Error messages generated during execution
    - Information about ID directories
    - Zone coverages
    - MAP directory summary.

* Purges the ID and MAP directories.

* Renames all files to have a file version of one (;1).

* Inserts copies of the ID files into CLEANUP.LOG.

* Checks for extra directories and reports the total number of directories.

* Deletes all DR_COVRG.DAT files.

* Attempts to perform open, read, scan, and seek operations on HEADER.DAT files. Reports the total number of HEADER.DAT files and error messages.

* Counts PA directories and insures that there is only one PA directory per zone.

* Verifies and reports the number of ROW directories.

* Counts CD directories and insures that the number of CD directories matches the number of CD-ROMS processed listed in the CHART STATUS file.

* Verifies and reports the number of LEGEND files.

* Verifies and reports the number SOURCE GRAPHICS files.

* Verifies and reports the number of PALETTE.DAT files and insures that each PA directory has a PALETTE.DAT file.

* Verifies and reports the number of COVERAGE.DAT files and insures that each PA directory has a COVERAGE.DAT file.

* Verifies and reports the number of SEGMENT files.

* For each zone, verifies that all segments fall within defined boundaries. Reports boundaries and any errors encountered.

* Checks for extraneous files.

3 Invocation

Use the following syntax to invoke CLEAN_UP.C:

        errors = CLEAN_UP ();

where errors is an INTEGER*4 (LONG) variable containing the number of errors found during execution. Detected errors may be corrected and CLEAN_UP re-invoked.

** CLEANUP should be invoked until an error count of zero is returned.

2 COORDINATES.FOR
 Contains routines that convert latitude/longitude coordinates into

F-59

their degrees, minutes, and seconds equivalents. Each routine is described as a subtopic.

3 DISPLAY_COORD
Double precision floating point latitude/longitude coordinate pairs are converted to their degrees, minutes and seconds equivalents and displayed on the user's terminal.

The following syntax is used to invoke DISPLAY_COORD:

       call DISPLAY_COORD(SOUTHLAT, WESTLONG, NORTHLAT, EASTLONG)

   where

       SOUTHLAT   Southern latitude coordinate
                     (passed, real*8)
       NORTHLAT   Northern latitude coordinate
                     (passed, real*8)
       WESTLONG   Western longitude coordinate
                     (passed, real*8)
       EASTLONG   Eastern longitude coordinate
                     (passed, real*8)

4 Example
The following FORTRAN-based pseudocode provides an example of usage:

```
C *** Using these coordinate values as input
      SOUTHLAT  = -25.25
      WESTLONG  = -125.75

      NORTHLAT  =  30.00
      EASTLONG  = -90.00

      CALL DISPLAY_COORD( SOUTHLAT, WESTLONG, NORTHLAT, EASTLONG)

C *** The following equivalent values are displayed:

  southlat:  -25.25 degrees =  -25 deg 15 min 00.00 sec
  westlong: -125.75 degrees = -125 deg 45 min 00.00 sec

  northlat:   30.00 degrees =   30 deg 00 min 00.00 sec
  eastlong:  -90.00 degrees =  -90 deg 00 min 00.00 sec
```

3 ENTER_COORD
Issues prompts for a pair of latitude/longitude coordinates. Because they are stored as a character string, the coordinates may be entered as either integer, decimal, or degrees, minutes and seconds values (i.e., 90, 90.25, or 90 15 00). When the coordinates are entered as integer or degrees, minutes, and

seconds, the character string is converted to a double precision floating point value by the function DECIMAL (see subtopic DECIMAL.FOR for additional information). When the coordinates are entered as decimal values, the character string is converted to its double precision floating point value by an internal READ statement.

The following syntax is used to invoke ENTER_COORD:

        call ENTER_COORD(NEW_LAT, NEW_LON)

    where
            NEW_LAT   Contains the converted latitude coordinate
                       (returned, real*8)
            NEW_LON   Contains the converted longitude coordinate
                       (returned, real*8)

2 CREATE_PA_DIR.C

            ---------- TO BE ADDED LATER ------------

2 DECIMAL.FOR
This function is designed to accept a character string from the routine ENTER_COORD.FOR and convert it to its double precision equivalent. The character string must contain a coordinate value in either decimal or degrees, minutes and seconds formats. Both formats are converted to a double precision floating point value.

The following syntax is used to invoke DECIMAL:

        REAL*8 FUNCTION DECIMAL (ATMP)
    where
        REAL*8   Returns the converted decimal value

        ATMP   Contains the coordinated to be converted
                  (passed, character*80)

3 Examples
The following FORTRAN-based pseudocode provides examples of usage:

C *** Using the following input string

    ATMP = '90 15 00'
    LAT = DECIMAL (ATMP)

C *** The returned value of LAT is 90.2500

C *** Using the following input string
    ATMP = '90'

```
      LAT = DECIMAL( ATMP)

 C*** The returned value of LAT is 90.0000
```

2 DTR_PROCS.FOR
 Contains a suite of FORTRAN language routines for converting to
 and from Datatrieve data storage format. Routines exist for
 converting to/from Datatrieve CHARACTER format to REAL*4 format.
 Subtopics provide descriptions of routines within the suite.

3 CLOSE_ADRG_DTR_FILE
 Closes an open (Datatrieve ADRG) file that has been designated as
 unit 10. CLOSE_ADRG_DTR_FILE is invoked using the following
 syntax:

```
        call CLOSE_ADRG_DTR_FILE
```

3 DECODE_DTR_LATLON
 Converts two longitude and two latitude values, from the
 Datatrieve "integer" format (which is actually in a character
 format), into REAL*4 format. DECODE_DTR_LATLON is invoked using
 the following syntax:

```
  call DECODE_DTR_LATLON (encoded,tlat,blat,llon,rlon)
```

  where
    STRUCTURE   encoded
                  (Passes the following fields)

Type	Name	Description
CHARACTER*12	serial	Chart serial number
CHARACTER*2	edition	Chart edition number
CHARACTER*4	llat	Chart minimum latitude
CHARACTER*4	ulat	Chart maximum latitude
CHARACTER*5	llon	Chart minimum longitude
CHARACTER*5	rlon	Chart maximum longitude
CHARACTER*29	geo_loc	Chart geographic location
CHARACTER*6	cdnum	ADRG CD serial number
CHARACTER*7	date	Date entered into database
CHARACTER*2	box	Backup box storing the ADRG CD
CHARACTER*1	cac	CAC inclusion flag
CHARACTER*1	dist	Distribution statement flag
CHARACTER*1	sheet	Flag for paper chart that is stored in the MDFF lab
REAL*4	blat	Minimum latitude of chart (returned)
REAL*4	tlat	Maximum latitude of chart (returned)
REAL*4	llon	Minimum longitude of chart (returned)
REAL*4	rlon	Maximum longitude of chart

(returned)

3 ENCODE_DTR_LATLON
Converts two longitude and two latitude values from REAL*4 format, into the Datatrieve "integer" format (which is actually in character format). ENCODE_DTR_LATLON is invoked using the following syntax:

    call ENCODE_DTR_LATLON (tlat,blat,llon,rlon,encoded)

where
    REAL*4   blat   Minimum latitude of chart
                          (passed)
    REAL*4   tlat   Maximum latitude of chart
                          (passed)
    REAL*4   llon   Minimum longitude of chart
                          (passed)
    REAL*4   rlon   Maximum longitude of chart
                          (passed)
    STRUCTURE   encoded
                (returns the following fields)
        CHARACTER*12   serial   Chart serial number
        CHARACTER*2   edition   Chart edition number
        CHARACTER*4   llat   Chart minimum latitude
        CHARACTER*4   ulat   Chart maximum latitude
        CHARACTER*5   llon   Chart minimum longitude
        CHARACTER*5   rlon   Chart maximum longitude
        CHARACTER*29   geo_loc   Chart geographic location
        CHARACTER*6   cdnum   ADRG cd serial number
        CHARACTER*7   date   Date entered into database
        CHARACTER*2   box   Backup box storing the ADRG CD
        CHARACTER*1   cac   CAC inclusion flag
        CHARACTER*1   dist   Distribution statement flag
        CHARACTER*1   sheet   Flag for paper chart that is
                                    stored in the MDFF lab

3 MODIFY_DTR_CHARTS_CACED
Reads charts in current CHART_STATUS file and changes the CAC coverage field in the appropriate ADRG database file from a blank to a 'C', which indicates which chart has been incorporated into a CAC. MODIFY_DTR_CHARTS_CACED is invoked using the following syntax:

        Call MODIFY_DTR_CHARTS_CACED()

3 OPEN_ADRG_DTR_FILE
Performs a formatted open (as unit 10) on a Datatrieve ADRG file. The ADRG file opened is based on the map scale. OPEN_ADRG_DTR_FILE is invoked using the following syntax:

```
        call OPEN_ADRG_DTR_FILE (map_scale)
```

where
    INTEGER*2 map_scale   ADRG map scale, ranges from 0 to 7
                               (passed)

3 READ_ADRG_DTR_RECORD

Returns a status indicating the existence of a record. If the
record exists, all fields, contained within the ADRG database are
returned. READ_ADRG_DTR_RECORD is invoked using the following
syntax:

```
        status =  READ_ADRG_DTR_RECORD(decoded)
```

where
  LOGICAL   status
             (returned)

STRUCTURE   decoded
            (returns the following fields)

Type	Field	Description
CHARACTER*12	serial	Chart serial number
CHARACTER*2	edition	Chart edition number
REAL*4	llat	Chart minimum latitude
REAL*4	ulat	Chart maximum latitude
REAL*4	llon	Chart minimum longitude
REAL*4	rlon	Chart max longitude
CHARACTER*29	geo_loc	Chart geographic location
CHARACTER*6	cdnum	ADRG CD serial number
CHARACTER*7	date	Date entered into database
CHARACTER*2	box	Backup box storing the ADRG CD
CHARACTER*1	cac	CAC inclusion flag
CHARACTER*1	dist	Distribution statement flag
CHARACTER*1	sheet	Flag for paper chart that is stored in the MDFF lab

2 FCOMMA.C

Places commas in an integer number. This function should only be
invoked by routines written in FORTRAN code. For the equivalent
C language function, see subtopic CCOMMA.C.

FCOMMA.C is invoked using the following syntax:

```
   character*(*) function FCOMMA (number)
```

where
       A character string, containing the number with newly
       inserted commas, is returned.

       number   Contains an integer number
                  (passed)

3 Example
The following pseudocode provides an example of usage.

```
        integer*4    number/100100999/          ! integer to be passed

        write (6, '(1x,a)') FCOMMA (number)   ! invocation

        ! The string "100,100,999" is printed.
```

2 FILE_ATTR_PROCS.C
Contains a suite of C language routines that provide information
about VMS files and directories. Subtopics include requirements
and individual routines within the suite.

3 Include_files
The file, "V2_DIR:F_FILE_ATTR.H", contains data definitions that
are used by suite routines and must be included in the source
code.

The following external routine is required for linkage:

F_FILE_ATTR.MAR   Implementation of the DCL lexical function
                    F$FILE_ATTRIBUTE as a "callable" subroutine

3 Suite_Description
FILE_ATTR_PROCS.C contains the following routines:

4 FILE_SIZE_INIT
Initializes the variables that are used by file_size_send and
file_size_ret. The following syntax is used to invoke
file_size_init:

```
        void file_size_init ()
```

4 FILE_SIZE_SEND
Determines whether the file, which is passed as an argument, is a
directory or a file. Calls a macro subroutine that reads the file
until end-of-file is encountered and determines the number of
actual blocks that the file, or directory uses. The following
syntax is used to invoke file_size_send:

```
        long file_size_send (struct dsc$descriptor_s *fname)
```

where
    long   Return status that will possess one of the following
                values: TRUE if no errors are encountered
                         FALSE if errors are encountered

     fname   Name of directory or file. Passed as
             struct dsc$descriptor_s

4 FILE_SIZE_RET
Returns the total number of blocks that are used by files and
directories which have been passed as arguments to the
file_size_send routine. The following syntax is used to invoke
file_size_ret:

void file_size_ret (long *f_blocks_used, long *d_blocks_used)

   where
        f_blocks_used   The total number of blocks used by the files.
                        (passed long)

        d_blocks_used   The total number of blocks used by the
                        directories.
                        (passed long)

3 Example
The following C language pseudocode provides an example of usage.

call FILE_SIZE_INIT ()   ! This sets up the variables used by the
                         ! next two calls. This routine MUST be
                         ! called first.

! Loop through the number of files you wish to obtain total sizes
! on.

   LOOP 1 TO Number_Of_Files

       ! Send in as many files as you like, may only be one.
       ! Filename is a string descriptor.

       call FILE_SIZE_SEND (filename)

   END LOOP

! This is the final routine and MUST be called last. It returns
! the total blocks used by files and directories, passed into
! FILE_SIZE_SEND. Arguments are passed by reference

call FILE_SIZE_RET (f_blocks_used, d_blocks_used)

2 FIND_CAC_BITMAP.C
Returns all CAC logged bitmap file names that are stored in the
MDFF_SYSTEM:[BITMAPS] directory for a given scale and zone.

FIND_CAC_BITMAP.C contains two routines, each is described as a
subtopic. In order for the process to work correctly, the
routines must be called in proper sequence.

3 Include_Files

The following file must be included for use with
FIND_CAC_BITMAP.C:

V2_DIR:DATA_DEFS.H   Contains type definitions for common
                      variables (e.g., scale, zone)

3 FIND_CAC_BITMAP_INIT

This routine must be used to pass the scale and zone of the
CAC bitmaps file names you wish to have returned by the other
routine, FIND_CAC_BITMAP_RET.

This routine MUST be invoked first. The following syntax is used
to invoke find_cac_bitmap_init:

```
void find_cac_bitmap_init (SCALE *scale, ZONE *zone)
```

where
```
      scale   INTEGER*4   Contains the chart scale
                          (passed, by reference)
      zone    INTEGER*4   Contains TS zone
                          (passed by reference)
```

3 FIND_CAC_BITMAP_RET

Returns CAC a bitmap file name. This routine should be invoked
iteratively until filename is returned NULL (i.e., no more files).

All CAC bitmaps will be returned using the scale and zone that
were passed earlier to the FIND_CAC_BITMAP_INIT routine.

The following syntax is used to invoke find_cac_bitmap_ret:

```
 char *find_cac_bitmap_ret (void)
```

where
```
      char      Contains a CAC bitmap file name
                (returned)

      void      There are no arguments passed
```

3 Example

The following (FORTRAN-like) code segment provides an example of
the correct usage.

```
 call FIND_CAC_BITMAP_INIT (scale, zone) ! Use this routine to
                                         ! pass the scale and zone of the
                                         ! CAC bitmaps file names you wish
                                         ! to have returned by the next
                                         ! call.
                                         ! This routine MUST be invoked
                                         ! first.
                                         ! Scale and zone are INTEGER*4
```

```
                                    ! and passed by reference.


     ! Loop through until filename is NULL (i.e., no more files).  All
     ! CAC bitmaps will be returned using the scale and zone that were
     ! passed to the FIND_CAC_BITMAP_INIT function.

     LOOP UNTIL filename = NULL
        filename = FIND_CAC_BITMAP_RET ()    ! Filename is a character
                                             ! string pointer and is
                                             ! returned
     END LOOP
```

2 FIND_FILE.FOR
Uses search and file names to find a file. This module contains
two entry points; one for calling from OpenVMS "C" (FIND_FILE_C)
and one for calling from OpenVMS FORTRAN (FIND_FILE).

Two entry points are required, due to the passing of strings by
descriptor in VMS FORTRAN. In the "C" version, string arguments
are of type "char *" (i.e., pointer to a NULL terminated string).
No include files required for invocation.

3 FORTRAN_Invocation
The following syntax is used in FORTRAN invocation:

           lstatus = FIND_FILE (searchname,reset,filename)

  where
           lstatus is a return value that contains one of the
           following values:
               When the search status is .TRUE. then a match was found.
               When the search status is .FALSE., no VMS filename
               matching the search specification was found.

           searchname: VMS filename specification search mask.
                       Wildcards (*) are allowed.
                       (passed, character*(*))

                reset: If .TRUE., this resets the search context.
                       This is used to set a new VMS filename
                       specification search string.
                       (passed, logical)

             filename: VMS filename specification matching the search
                       mask.
                       (returned, character*(*))
3 C_Invocation
The following syntax is used in C invocation:

                              F-68
```

```
        lstatus = FIND_FILE_C (searchname,reset,filename)
```

where

        lstatus is a return value that contains one of the
        following values:
            When the search status is .TRUE. then a match was found.
            When the search status is .FALSE., no VMS filename
            matching the search specification was found.

        searchname: VMS filename specification search mask.
                    Wildcards (*) are allowed.
                    (passed, char*)

            reset: If .TRUE., this resets the search context.
                    This is used to set a new VMS filename
                    specification search string.
                    (passed, char*)

        filename: VMS filename specification matching the
                    search mask.
                    (returned, char*)

2 GENERIC_QUEUE_STOPPED.FOR

A FORTRAN function that determines whether a generic queue,
defined by the currently set processing thread, is stopped.
A status is returned with one of the following values:

        returns TRUE   when the generic queue is stopped
        returns FALSE   when the generic queue is NOT stopped

3 Invocation
The following syntax is required for proper invocation.

        lstatus = GENERIC_QUEUE_STOPPED ()

where

        lstatus is the return status of the generic queue, that
        is declared as a LOGICAL variable.

    returns TRUE   when the generic queue is stopped.
    returns FALSE   when the generic queue is NOT stopped.

3 Example
The following example uses FORTRAN-based pseudocode to show
proper use of GENERIC_QUEUE_STOPPED.

    logical   generic_queue_stopped   ! The function declaration

F-69

```
      logical    lstatus                    ! The status returned by
                                             ! generic_queue_stopped

      c****************************************************************
      c*  Returns .TRUE. if queue is stopped.                        *
      c*  NOTE:  Set the processing thread to be tested before running*
      c****************************************************************

        lstatus = GENERIC_QUEUE_STOPPED ()

      c* Print status of Generic Queue *
          if (lstatus .eq. .true.) then
               print*,' '
               print*, 'Generic Queue Stopped...'
             else
               print*,' '
               print*, 'Generic Queue NOT Stopped...'
          endif
```

2 GET_DS_SEGMENT_NAME.FOR

This subroutine builds the complete downsampled segment path name
(excluding the color/zone file extension (e.g., ".R_NT").

Two types of root directory paths (i.e., "dspath" argument) for
the downsampled data are supported:

>   Root directory paths terminating in a colon   ":"
>   Root directory paths terminating in a bracket "]"

When the downsampled segment root directory path terminates in a
":" the complete segment path name (i.e., the "segpath" argument)
is completed by adding the string:

$$\text{"[Rsnnnnn]12345678"}$$

When the downsampled segment root directory path terminates in a
"]" the complete segment path name (i.e., the "segpath" argument)
is completed by removing the "]" and adding the string:

$$\text{".Rsnnnnn]12345678"}$$

See topic CAC_PROCESSING, subtopic Segment_Files for documentation
on downsampled segment file naming conventions.

3 Invocation
Subroutine GET_DS_SEGMENT_NAME should be invoked using the
following syntax:

```
        call GET_DS_SEGMENT_NAME (dspath,row,col,segpath,seglen)
```

Items contained within the parameter list include:

    dspath: Root destination directory for the downsampled data
        For example, CHART_SEGMENTS:.
        (passed, character*(*))

    row: Row number of downsampled segment data.
        (passed, integer*4)

    column: Column number of downsampled segment data.
        (passed, integer*4)

    segpath: Complete downsampled segment path name (excluding
        the color/zone file extension: .R_NT for example).
        (returned, character*(*))

    seglen: Length of "segpath"
        (returned, integer*4)

3 Required_Subroutines
The following subroutines are utilized by GET_DS_SEGMENT_NAME:

    ENCODE_KEY: Create the segment key name from the row and column.
    STRING_LENGTH: Find the length of a string. Terminates on NULL characters, blank characters and the size of storage allocated for the string.

No "INCLUDE" (i.e., external) files are required.

2 GET_ODI_DATA.C
GET_ODI_DATA contains routines that read mapstation subdirectories, extract row and column values for each of the segments contained within the subdirectories, and store the sorted row/column values in a structure.

GET_ODI_DATA contains the following routines, each of which is described as a subtopic.

    GET_ODI_DATA   Fills the structure containing row and column values. Returns the number of segments found, the filled structure and a status flag.

    QUICKSORT   Performs a quicksort, by ascending row, on the passed row/column structure.

    GET_HEX_DIGITS   Is passed a number (stored in a byte) and uses

masks to obtain the number's hexadecimal equivalent.

3 GET_ODI_DATA

GET_ODI_DATA is passed the row/column structure that will contain row/column values for all segments contained within the specified subdirectories, and the number of subdirectories to be included.

GET_ODI_DATA prompts the user for mapstation subdirectory filenames, which are opened one at a time. The first 4 bytes of subdirectory files are read to obtain information about the subdirectory - mainly the number of keynames that are contained within the subdirectory.

Keynames are extracted and decoded to their row/column values and then placed in the row/column structure.

A count of the number of segments contained within the subdirectory is kept. This count is returned as a long variable.

Once all subdirectory filenames are entered, the operator terminates input by entering 'EXIT'.

The newly filled structure, containing row and column values, is sent to a quicksort where the row and column pairs are sorted in ascending row number.

For all segments contained within the mapstation subdirectories, GET_ODI_DATA returns the newly filled structure containing sorted row and column values, a status flag indicating termination status and the total number of segments found.

4 Invocation

GET_ODI_DATA is invoked using the following syntax.

```
long get_odi_data (struct coord ROW_COL [SEG_MAX], long *flag,
                   *long num_files)
```
where

long	A variable containing the number of segments, within the subdirectories, that were found
struct coord ROW_COL [SEG_MAX]	Array containing the structures for row/column values
long flag	Status return flag. Normal return status = FALSE Abnormal return status = TRUE
long num_files	Passed value represents the number of mapstation sub-directories to be read.

Returned value represents the total number of segment files found.

3 QUICKSORT

Performs a quicksort, by ascending row, on the passed row/column structure.

4 Invocation

QUICKSORT is invoked using the following syntax.

void quicksort (struct coord ITEM[], long left, long right)

where
struct coord ITEM[]	Array structure containing row/column pairs
long left	A pointer to the structure's first element
long right	A pointer to the structure's last element

3 GET_HEX_DIGITS

When passed a number stored in a byte, GET_HEX_DIGITS uses masks to obtain the hexadecimal equivalent.

For example:
The binary number 01010010 equivalent in hexadecimal is 52.

The 5 (i.e., from the left-most 4 bits) is returned in the variable named LEFT_HEX.

The 2 (i.e., from the right-most 4 bits) is returned in the variable named RIGHT_HEX.

4 Invocation

GET_HEX_DIGITS is invoked using the following syntax.

void GET_HEX_DIGITS (unsigned *right_hex, unsigned *left_hex,
                    unsigned number)

where
unsigned *right_hex	Right-most digit of hexadecimal equivalent. A return value.
unsigned *left_hex	Left-most digit of hexadecimal equivalent. A return value.
unsigned number	Input value to be converted to hexadecimal

3 Required_Subroutines

The following routine is utilized by GET_ODI_DATA:

F-73

DECODE_KEY.FOR   A FORTRAN subroutine that decodes a key name
                  into its row and column components. For
                  additional information see CAC_Source_Code
                  subtopic DECODE_KEY.FOR

2 GET_PA_SEGMENT_NAME.FOR
This subroutine builds the complete compressed segment path name
(including the zone number in the file extension).

Two types of root directory paths (i.e., "papath" argument) for
the compressed data are supported. One root directory path
terminates with a colon ":" and the other terminates with a "]".

When the compressed segment path terminates with a ":" the
complete segment path name (i.e., "segpath" argument) is completed
by adding the string:

            "[Rsnnnnn]12345678.90.z"

When the compressed segment path terminates with a "]" the
complete segment path name (the "segpath" argument) is completed
by removing the "]" and adding the string:

            ".Rsnnnnn]12345678.90z"

See topic CAC_PROCESSING, subtopic Segment_Files for documentation
on compressed segment file naming conventions.

3 Invocation
Subroutine GET_PA_SEGMENT_NAME should be invoked using the
following syntax:

    call GET_PA_SEGMENT_NAME (papath,row,col,segpath,seglen)

Items contained within the parameter list include:

    papath: Root destination directory for the compressed data
            for example, CHART_ODI_DISK:[MAP5.PA013301].
            (passed, character*(*))

       row: Row number of compressed segment data.
            (passed, integer*4)

    column: Column number of compressed segment data.
            (passed, integer*4)

   segpath: Complete compressed segment path name, including the
            zone number in the file extension.
            (returned, character*(*))

```
seglen: Length of "segpath"
          (returned, integer*4)
```

3 Required_Subroutines
  The following subroutines are utilized by GET_PA_SEGMENT_NAME:

    ENCODE_KEY: Create the segment key name from the row and
                column.

    STRING_LENGTH: Find the length of a string. Terminates on
                   NULL characters, blank characters, and the size
                   of storage allocated for the string.

  No "INCLUDE" (i.e., external) files are required.

2 GET_PID.FOR
  Contains routines that obtain a Process Identification (PID)
  value. Each routine is described as a subtopic.

3 GET_PID
  This routine uses VAX/VMS system services to obtain the calling
  process's PID. The PID is returned in numeric (hexadecimal)
  format.   The concept of PIDs is unique to VAX/VMS. Hence, this
  routine may not be applicable to other operating systems. No
  include files are required for invocation.

  The following syntax is used to invoke GET_PID:

        call GET_PID (pid)
    where
        pid: Current process's Process Identification Code. This
             is a hexadecimal number. For additional information,
             see PID in the VAX/VMS documentation.
             (returned, integer*4)

3 GETPID_ASC
  This routine uses VAX/VMS system services to obtain the calling
  process's PID. The numeric PID is translated into a character
  string and returned in text (ASCII) format. PIDs are normally
  eight characters long. The concept of PIDs is unique to VAX/VMS.
  Hence, this routine may not be applicable to other operating
  systems. No include files are required for invocation.

  The following syntax is used to invoke GET_ASC:

    call GETPID_ASC (pid_string)
    where
    pid_string: The calling process's VAX/VMS PID in text format.
                (returned, character*(*))
```

2 GET_UNIX_BINARY_TIME.C

GET_UNIX_BINARY_TIME.C contains a suite of C language functions that are used to convert a VAX time string into the corresponding UNIX binary time.

Functions contained within the suite are described as subtopics.

3 GET_UNIX_BINARY_TIME

When passed a VAX time string, this subroutine calls functions to convert the time string to uppercase, get UNIX binary time as a VAX quadword, and convert that quadword to a longword. The UNIX binary time is returned.

GET_UNIX_BINARY_TIME is invoked using the following syntax:

        long GET_UNIX_BINARY_TIME (char *timestr)
where
        long   the return value, UNIX binary time
        char *timestr   the VAX time string, passed as input

3 CONVERT_QUAD_TO_LONG

When passed a VAX quadword as two longwords containing a UNIX binary time in nanoseconds, this function returns the UNIX binary time in seconds as a longword.

CONVERT_QUAD_TO_LONG is invoked using the following syntax:

        long CONVERT_QUAD_TO_LONG (long one, long two)
where
        long   the return value, UNIX binary time
        long one   first part of the VAX quadword, passed value
        long two   second part of the VAX quadword, passed value

3 UPPER_CASE

Converts a character string to uppercase.

UPPER_CASE is invoked using the following syntax:

        void UPPER_CASE (char str [], long str_length)
where
        char str   character string being converted to upper case
    long str_length   length of str[]

3 Required_Subroutines

TIMEM.MAR, is required for program linkage. See CAC_SOURCE_CODE subtopic TIMEM.MAR for additional information.

2 IIS_PLOT_PROCS.FOR

IIS_PLOT_PROCS contains a suite of FORTRAN subroutines and functions that enable the use of graphic devices and generation of hardcopy plots on the TEKTRONIX printer. The display image is written into a device independent buffer that can be used with any graphics and/or image device (e.g., IVAS, VWS color monitors).

All routine names begin with the string "IIS_" followed by a simple function descriptor. For example, the routine named IIS_LABEL is used for labeling purposes. There are routines for displaying 8-bit and 24-bit data. The 8-bit routines are differentiated from the 24-bit routines by adding "8_BIT" to the routine's name. For example, the routine IIS_LABEL is used for labeling 24-bit images and the routine IIS_LABEL_8BIT is used for labeling 8-bit images.

The subroutines and functions contained within IIS_PLOT_PROCS are described as subtopics. Information concerning proper usage, invocation and parameters is included within the subtopics.

For retrieval purposes within MDFFHELP, routines IIS_BORDER. and IIS_LABEL. are terminated with a period - this is done only to differentiate them from other routines with similar names.

3 IIS_IVAS_INIT

Function IIS_IVAS_INIT contains IVAS commands necessary to allocate the IVAS 600 for stand-alone operation. This function must be invoked by all programs using the IVAS and be one of the first routines invoked.

4 Input_Output

Parameters for IIS_IVAS_INIT include the following:

  Input parameters: None

  Output parameters: None

       Return value: INTEGER*4, channel opened by the FIVASOPEN
                     call. This value is needed to close the channel
                     in order to release the device upon termination
                     of a program.

If the channel is not closed (using FIVASCLOSE(ivaschan)), the device may remain allocated to the user, not allowing others access to the IVAS.

4 Invocation

IIS_IVAS_INIT is invoked using the following syntax:

        ivaschan = IIS_IVAS_INIT()

3 IIS_8BIT_INIT
Function IIS_8BIT_INIT is used to load the 8-bit color definitions
into the 8-bit color table. IIS_8BIT_INIT must be invoked before
loading the color table. The color definitions are added into the
table starting at index 241.

4 Input_Output
Parameters for IIS_8BIT_INIT include the following:

 Input parameters:

     byte   RGBTABLE (256,3)   A 256 by 3 array of RGB colors.

Output parameters: None

4 Invocation
IIS_8BIT_INIT is invoked using the following syntax:

     call IIS_8BIT_INIT (rgbtable)

3 IIS_LABEL
Subroutine IIS_LABEL is used to write 24-bit text onto a graphics
monitor (i.e., any 24-bit color monitor, such as the IVAS color
monitor). Text is written using the red, green, and blue buffers
in the user's program. Text appears on the monitor and on any hard
copies that are produced on the TEKTRONIX printer.

Because IIS_LABEL loads the image buffer with desired text,
IIS_LABEL must be invoked prior to loading the buffer onto the
IVAS color monitor (via appropriate subroutine). Similarly,
IIS_LABEL must be invoked prior to creating a TEKTRONIX print file
(via subroutine CREATE_IVAS_PRINTEK_IMAGES).

4 Character_Set
Printable characters are listed below.

Upper and lower case characters:

        ABCDEFGHIJKLMNOPQRSTUVWXYZ
        abcdefghijklmnopqrstuvwxyz

Numbers:
        0123456789

Other characters:

    + - = ( ) * & % $ # @ , . | \ / ? < > ! ' " [ ]

4 Input_Output
The following parameters are required as input:

character*(*)TLABEL   A string of currently defined characters, of a length assigned in the calling routine, to write into the image.

integer*4   XOFF   Number of pixels from the left side of image to place the start of "TLABEL".

integer*4   YOFF   Number of pixels from the top of image to place the start of "TLABEL".

XOFF and YOFF refer to the placement of the lower left corner of the first character of "TLABLE"

character*1 TDIRECT   Defines whether the string "TLABEL" is to be written horizontally (input value 'H') or vertically (input value 'V').

integer*2   ULINE   Underline option. If "ULINE" is 0, then the label "TLABEL" is not underlined. A value of 1 will place a single line under "TLABEL" with a single pixel gap between the characters and the line. With a value of 2, two lines are placed under the characters with no gap between the line and the characters.

logical   ON_BLACK   If true, then the characters are written into the image on a black background, overwriting image data. If false, then characters are written with the image data as background (i.e., image data is not blacked out).

integer*2   RGB   Used for defining text color. This value is a hexadecimal value with the following values:

'rgb'x: where rgb refers to the red, green, and blue guns of the IVAS. A value of 0 for r, g, or b indicates the gun is "off," a value of 1 indicates the gun is "on."

'rgb' values	resultant colors
'000'x	BLACK (not recommended)
'100'x	RED
'010'x	GREEN
'001'x	BLUE
'110'x	YELLOW
'011'x	CYAN
'101'x	MAGENTA
'111'x	WHITE

F-79

```
     byte RBUF(XDIM,YDIM)  Two-dimensional byte buffer containing
                           the red component of the image.

     byte GBUF(XDIM,YDIM)  Two-dimensional byte buffer containing
                           the green component of the image.

     byte BBUF(XDIM,YDIM)  Two-dimensional byte buffer containing
                           the blue component of the image.

     integer*4     XDIM    First dimension of RBUF, GBUF and BBUF
                           byte array; represents the number
                           of pixel columns in the image.

     integer*4     YDIM    Second dimension of RBUF, GBUF and BBUF
                           byte arrays; represents the number
                           of pixel rows in the image.
```

Output parameters: None

4 Invocation
     IIS_LABEL is invoked using the following syntax

          call IIS_LABEL (tlabel, xoff, yoff, tdirect, uline, rgb,
     *   rbuf, gbuf, bbuf, xdim, ydim)

3 IIS_LABEL_8BIT
 Subroutine IIS_LABEL_8BIT is used to write 8-bit text onto a
 graphics monitor (i.e., any 8-bit color monitor). Text is written
 using one buffer.

 For printing on the TEKTRONIX printer, the single 8-bit buffer is
 converted into 8-bit red, green, and blue buffers (resulting with
 a 24-bit image). Since the 8-bit value of the characters are used
 as an index into the color table, the conversion to 24-bit still
 yields the identical color defined (i.e., the 8-bit red value is
 converted to its 24-bit equivalent).

 Because IIS_LABEL_8BIT loads the image buffer with desired text,
 IIS_LABEL_8BIT must be invoked prior to loading the buffer onto
 the IVAS color monitor (via appropriate subroutine). Similarly,
 IIS_LABEL_8BIT must be invoked prior to creating a TEKTRONIX print
 file via subroutine CREATE_IVAS_PRINTEK_IMAGES.

4 Character_Set
  Printable characters are listed below.

  Upper and lower case characters:

            ABCDEFGHIJKLMNOPQRSTUVWXYZ
            abcdefghijklmnopqrstuvwxyz

                          F-80
```

Numbers:

0123456789

Other characters:

+ - = ( ) * &  % $ # @ , . | \ / ? < > ! ' " [ ]

## 4 Input_Parameters

The following parameters are required as input:

character*(*)TLABEL   A string of currently defined characters, of a length assigned in the calling routine, to write into the image.

integer*4      XOFF   Number of pixels from the left side of image to place the start of "TLABEL".

integer*4      YOFF   Number of pixels from the top of image to place the start of "TLABEL".

XOFF and YOFF refer to the placement of the lower left corner of the first character of "TLABLE".

character*1 TDIRECT   Defines whether the string "TLABEL" is to be written horizontally (input value 'H') or vertically (input value 'V').

integer*2      ULINE   Underline option. If "ULINE" is 0, then the label "TLABEL" is not underlined. A value of 1 will place a single line under "TLABEL" with a single pixel gap between the characters and the line. With a value of 2, two lines are placed under the characters with no gap  between the line and the characters.

logical    ON_BLACK   If true, then the characters are written into the image on a black background, overwriting image data. If false, then characters are written with the image data as background (i.e., image data is not blacked out).

integer*2      RGB   Used for defining text color. This value is a hexadecimal value with the following values:

'rgb'x: where rgb refers to the red, green, and blue guns of the graphics device. A value of 0 for r, g, or b indicates the gun is "off",

a value of 1 indicates the gun is "on".

| 'rgb' values | resultant colors |
|---|---|
| '000'x | BLACK (not recommended) |
| '100'x | RED |
| '010'x | GREEN |
| '001'x | BLUE |
| '110'x | YELLOW |
| '011'x | CYAN |
| '101'x | MAGENTA |
| '111'x | WHITE |

byte RGBBUF(XDIM,YDIM)  Two-dimensional byte buffer containing the red, green, and blue components of the image.

integer*4    XDIM  First dimension of RGBBUF byte array; represents the number of pixel columns in the image.

integer*4    YDIM  Second dimension of RGBBUF byte array; represents the number of pixel rows in the image.

Output parameters: None

4 Invocation
IIS_LABEL_8BIT is invoked using the following syntax:

```
    call IIS_LABEL_8BIT (tlabel, xoff, yoff, tdirect, uline, rgb,
   *                     rgbbuf, xdim, ydim)
```

3 IIS_BORDER
Subroutine IIS_BORDER places a border around a 24-bit image.

4 Input_Output
The following parameters are required as input:

integer*2   RGB Used for defining border color. This value is a hexadecimal value with the following values:

'rgb'x: where rgb refers to the red, green, and blue guns of the graphics device. A value of 0 for r, g, or b indicates the gun is "off", a value of 1 indicates the gun is "on".

'rgb' values   resultant colors

```
                ----------------------------------
                '000'x              BLACK (not recommended)
                '100'x              RED
                '010'x              GREEN
                '001'x              BLUE
                '110'x              YELLOW
                '011'x              CYAN
                '101'x              MAGENTA
                '111'x              WHITE
```

integer*2 THICKNESS   Indicates the number of lines used in
                        drawing border. An increasing number of
                        lines increases the border thickness.
                        Allowed values: 1, 2, 3.
                        A value less than 1 is set to 1 and
                        a value greater than 3 is set to 3.

byte   RBUF   A two-dimensional byte buffer containing
                        the red component of the image.

byte   GBUF   A two-dimensional byte buffer containing
                        the green component of the image.

byte   BBUF   A two-dimensional byte buffer containing
                        the blue component of the image.

integer*4   XDIM   First dimension of RBUF, GBUF, and BBUF
                        byte arrays; represents the number of
                        pixel columns in the image.

integer*4   YDIM   Second dimension of RBUF, GBUF, and BBUF
                        byte arrays; represents the number of
                        pixel rows in the image.

Output parameters: None

4 Invocation
 IIS_BORDER is invoked using the following syntax:

     call IIS_BORDER (rgb, thickness, rbuf, gbuf, bbuf, xdim, ydim)

3 IIS_BORDER_8BIT
 Subroutine IIS_BORDER_8BIT is used to place a border around an
 8-bit image for display on a graphics device or on TEKTRONIX hard
 copies.

4 Input_Output
 The following parameters are required as input:

     integer*2   RGB   Used for defining border color. This value

                            F-83
```

is a hexadecimal value with the following values:

'rgb'x: where rgb refers to the red, green, and blue guns of the graphics device. A value of 0 for r, g, or b indicates the gun is "off", a value of 1 indicates the gun is "on".

'rgb' values	resultant colors
'000'x	BLACK (not recommended)
'100'x	RED
'010'x	GREEN
'001'x	BLUE
'110'x	YELLOW
'011'x	CYAN
'101'x	MAGENTA
'111'x	WHITE

integer*2	THICKNESS	Indicates the number of lines used in drawing border. An increasing number of lines increases the border thickness. Allowed values: 1, 2, 3. A value less than 1 is set to 1 and a value greater than 3 is set to 3.
byte	RGBBUF	Two-dimensional byte buffer containing the red, green, and blue components of the image.
integer*4	XDIM	First dimension of RGBBUF byte array; represents the number of pixel columns in the image.
integer*4	YDIM	Second dimension of RGBBUF byte array; represents the number of pixel rows in the image.

Output parameters: None

4 Invocation

IIS_BORDER_8BIT is invoked using the following syntax:

```
    call IIS_BORDER_8BIT (rgb, thickness, rgbbuf, xdim, ydim)
```

3 IIS_PLOT_DRIVER
Program IIS_PLOT_DRIVER is a program that displays the currently defined character set onto the IVAS monitor and produces a hard copy on the TEKTRONIX plotter. The character set is displayed several times, once in each of the definable colors. The user has

the option of using an 8-bit or 24-bit display (i.e., invocation of subroutines IIS_LABEL or IIS_LABEL_8BIT).

A hard copy can be produced by entering a file name at the prompt. Entering a <CTRL_Z> at the prompt will terminate program execution without producing a hard copy.

IIS_PLOT_DRIVER can be executed from any node. However, only node "AVENGR" is capable of both display on the IVAS color monitor and generating a hard copy on the TEKTRONIX plotter. All other nodes may only generate hard copies on the TEKTRONIX plotter.

3 IIS_PLOT_PROCS.INC

Common to each routine within IIS_PLOT_PROCS suite is the INCLUDE file IIS_PLOT_PROCS.INC. This file contains the character set definitions.

IIS_PLOT_PROCS.INC is not necessary unless access to variables names, which are used to display the defined character set, is required. Hence, the program IIS_PLOT_DRIVER uses this file (and for no other reason).

Variable names containing the defined character set can be found in either IIS_PLOT_DRIVER.FOR source code or IIS_PLOT_PROCS.INC source code.

2 LL_MAXMIN.C

LL_MAXMIN.C contains routines that determine the true pair of minimum and maximum coordinates from a set of coordinate pairs. These coordinate pairs can be expressed as row/column values or latitude/longitude values. Also, LL_MAXMIN determines whether the set of coordinate pairs transition 180 or 0 degrees longitude and adjusts the minimum and maximum coordinates accordingly.

3 Routines

LL_MAXMIN consists of four routines, only three will be used at any given time. Each routine is described below.

LL_MAXMIN_INIT: Initializes the variables used by the other subroutines. This routine MUST be called first.

LL_MAXMIN_SEND_RC: Repeat for each row/column pair within the set. This is invoked if the calling routine has row/column values available. If this routine is called, LL_MAXMIN_SEND is not used.

LL_MAXMIN_SEND: Repeat for each latitude/longitude pair within

F-85

the set. This is called if the calling
routine has latitude/longitude values
available. If this routine is called,
LL_MAXMIN_SEND_RC is not used.

LL_MAXMIN_RET: The final subroutine, must be called last.
Returns the minimum and maximum coordinates
(in terms of LATITUDE and LONGITUDE) from all
of the pairs passed to LL_MAXMIN_SEND.

In order to work correctly, the three selected subroutines MUST be
invoked in proper sequence. The Example subtopic shows proper
usage.

3 Example
The following example uses pseudocode to show proper use of
LL_MAXMIN and its associated subroutines.

```
call LL_MAXMIN_INIT (scale, zone) ! Initialize the variables used
                                  ! by the next two calls.  Scale
                                  ! and zone are INTEGER*4
                                  ! (or longs) and are passed
                                  ! by reference.  This routine
                                  ! MUST be called first.

! Loop - repeat for each latitude/longitude or row/col pair within

! the set.
! If the calling program has row/col values, use LL_MAXMIN_SEND_RC

! If the program has latitude/longitude values, use
LL_MAXMIN_SEND

     LOOP 1 to Number_Of_Coordinate_Pairs

          ! Send as many latitude/longitude pairs as desired.
          ! Latitude and Longitude are REAL*8 (or DOUBLE) and
          ! are passed by reference.

               call LL_MAXMIN_SEND (latitude, longitude)


               ..... OR ....


          ! Send as many row/col pairs as desired.
          ! Row and Column values are INTEGER*4 (or longs) and
          ! are passed by reference.

               call LL_MAXMIN_SEND_RC (row, col)
```

F-86

```
        END LOOP

! This is the final subroutine and MUST be called last.  It
! returns the minimum and maximum coordinates (latitude and
! longitude) from all of the pairs passed to LL_MAXMIN_SEND
! or to LL_MAXMIN_SEND_RC.
! Normal minimum and maximum values are used when the pairs
! have been determined to ONLY cross 0 degrees longitude.
! However, when the set crosses 180 degrees, or both 0 and 180
! degrees, the least positive and least negative longitude
! coordinates are used for the maximum longitude and minimum
! longitude values, respectively.
! Latitude and Longitude are REAL*8 (or DOUBLE) and are passed
! by reference.

        call LL_MAXMIN_RET (minlon, maxlon, minlat, maxlat)
```

2 MAP_DIR_PROCS.C

MAP_DIR_PROCS.C contains a suite of C language routines that
perform a variety of operations on a MAP directory. Subtopics
include requirements, individual routines within the suite
and subroutines that are external to the suite.

3 External_Subroutines
The following external routines are required for linking:

```
          GET_PA_DIR.FOR: Gets the PA directory name
          GETPID_ASC.FOR: Gets the process' system ID number.
       CHART_STATUS.FOR: Reads the CHART_STATUS file.
 SEND_TO_MDFF_CONSOLE.MAR: Send messages to display on the MDFF
                           operator console(s).
          FINDZONE.FOR: Determines which TS zone the data are
                           in.
```

3 Requirements
The following subtopics describe items that are required by
the routine suite.

4 Arrays
Some routines require array variables - the array names and a
list of their elements include:

Array COORD[4] defines an area by 2 corners in latitude/longitude
where:
```
        COORD[0] = southwest lat;
        COORD[1] = southwest lon;
        COORD[2] = northeast lat;
        COORD[3] = northeast lon;
```

Array CORNERS[4] defines an area by 2 corners in row/column

F-87

where:
```
     CORNERS[0] = southwest row;
     CORNERS[1] = southwest col;
     CORNERS[2] = northeast row;
     CORNERS[3] = northeast col;
```

Array LIMITS[2] defines upper and lower boundary expressed in latitude where:
```
     LIMITS[0] = bottom lat;
     LIMITS[1] = top lat;
```

4 Include_Files

The file "V2_DIR:MAP_DIR_PROCS.H" contains data definitions used by the routines and must be included in the source code.

4 Global_Variables

Global variables, which are to be declared external (i.e., extern), include the following:

```
     long    COL_LIMIT;   Number of columns in a given zone

     double LIMITS[2];   Described above

   double TOP_LAT, BOT_LAT;   Top and bottom latitudes of a zone

     char FORM[MAX_LEN];   Form = "CHART_SEGMENTS" for DS Data
                           Form = "MAP" for COMPRESSED Data

   char TRIMMED_DIR[MAX_LEN]; Trimmed Directory name
```

3 Suite_Descriptions

Individual routines are described as subtopics.

4 BUILD_STRUCT

```
void BUILD_STRUCT (long row, long col, struct coord ITEM [], long
                   *count);
```
Is passed a row and a column and places them into the structure COORD.

4 CAL_ZONE_COVERAGE

```
double CAL_ZONE_COVERAGE (long zone, long scale);
```

Is passed the zone and scale.

The global array LIMITS[2], is filled with the upper and lower latitude values for that zone.

Returns the column width in segments, COL_LIMIT.

4 CORNER_IN_BOUNDS

```
long CORNER_IN_BOUNDS (long zone, long scale, long row, long col,
```

```
                    double coord[]);
```

Is passed the zone, scale, row/column to be tested and
bounds.

Returns   TRUE if the segment is within bounds
          FALSE if the segment is not within bounds.

4 DECIDE_TO_DEL_NONPOLAR
  long DECIDE_TO_DEL_NONPOLAR (long corners[], long row, long col);

Is passed row/column values and determines whether or not
the nonpolar segment should be deleted.

Returns   TRUE if the segment should be deleted
          FALSE if the segment should not be deleted

4 DECIDE_TO_DEL_POLAR
  long DECIDE_TO_DEL_POLAR (long zone, long scale, long row,
                            long col, double coord[]);

Is passed the zone, scale, row/column to be tested, and
bounds.

Returns   TRUE if the segment is within polar bounds
          FALSE if the segment is not within polar bounds.

4 DELETE_DIR
  void DELETE_DIR (char *buffer);

Sets the protection level of a file to DELETE and then
deletes it.

4 DEL_OR_SAVE_SEGMENT
  long DEL_OR_SAVE_SEGMENT (long zone, long scale, long row,
                            long col,long corners[], double
                            coord[]);

Calls functions to determine whether or not a segment
should be deleted.

Returns TRUE if the segment should be deleted
        FALSE if the segment should not be deleted

4 DETERMINE_BOUNDS
  long DETERMINE_BOUNDS (double coord[]);

Adjusts the array COORD to take into account zone overlap.

Returns TRUE if COORD was modified.

FALSE if COORD was not modified.

4 ELIMINATE
  void ELIMINATE (char *path, char *eliminate);

      When passed a path name, eliminates the substring
      *eliminate.

4 GET_CORNERS
  void GET_CORNERS (long scale, long zone, long *corners, double
                    *coord);]

      Is passed a chart scale, zone, and coordinates.

      Returns row/column corners for the segment.

4 GET_CS_TRIMMED_DIR
  long GET_CS_TRIMMED_DIR ();

      Gets the COMPRESSED data trimmed directory name from the
      processing thread and stores it in the global variable,
      TRIMMED_DIR.

4 GET_DS_DIR
  long GET_DS_DIR (char fname[]);

      Finds the MAP Directory of the current processing thread,
      extracts the scale, and returns a path name to the map
      directory as fname.

4 GET_DS_TRIMMED_DIR
  long GET_DS_TRIMMED_DIR ();

      Gets the DS data trimmed directory name from the processing
      thread and stores it in the global variable, TRIMMED_DIR.

4 GET_KEY_FROM_DS_FILENAME
  long GET_KEY_FROM_DS_FILENAME (struct dsc$descriptor_s fname,
                                 long *zone);

      Is passed a downsampled segment file name, as fname,
      and zone.

      Extracts and returns the segment file's key name.

4 GET_KEY_FROM_FNAME
  long GET_KEY_FROM_FNAME (struct dsc$descriptor_s fname);

      Is passed a key name as fname.
      Returns the key fname as (converted to) a ten digit
      integer.

F-90

4 GET_LOGICAL_NAME
long GET_LOGICAL_NAME (char *logical, char *resultant);

>   Is passed a logical name and returns its equivalence.

>   Returns   TRUE if an equivalence is found
>   FALSE if an equivalence is not found

4 GET_MAP_DIR
long GET_MAP_DIR (char fname[]);

>   Finds the MAP Directory of the current processing thread,
>   extracts the scale, and returns a path name to the map
>   directory.

4 GET_PA_NUM
long GET_PA_NUM (char *string);

>   Extracts a PA number for a PA directory.

4 GET_ROW_FROM_FNAME
long GET_ROW_FROM_FNAME (struct dsc$descriptor_s);

>   When passed a row name as FNAME, the row name is
>   converted to a six digit integer and returned.

>   When the six digit row number begins with a 1,
>   the 1 is dropped and the row number is made negative.

>   A value of 999999 is returned when the fname is not
>   a row fname.

4 LATLON_IN_CORNER
long LATLON_IN_CORNER (long scale, long zone, long row, long col,
                              double lat, double lon);

>   Is passed the scale, zone, row/column to be tested,
>   and the lat/lon of a corner.

>   Returns   TRUE if the segment is a corner
>   FALSE if the segment is not a corner

4 MAKE_DIR_FILE
void MAKE_DIR_FILE (char *buffer);

>   Is passed a string containing a file name in the form
>   (example) CHART_ODI_DISK:[MAP3.PA013001.R000000]TEMP.DOC;

>   The ']' bracket is removed and replaced with '.DIR'
>   to form a directory file name.

F-91

The last '.' period is replaced with a ']' bracket
before the new directory file name.

The final result is

CHART_ODI_DISK:[MAP3.PA013001]R000000.DIR;

4 MAKE_FILE_DIR
struct dsc$descriptor_s MAKE_FILE_DIR (struct dsc$descriptor_s);

Is passed a string descriptor containing a file name in
the form CHART_ODI_DISK:[MAP3.PA013001]R000000.DIR;1

The end ']' bracket is removed and replaced with a '.'
period.

The last '.' period is replaced with an end bracket,
the extension and version numbers are removed.

The new file name descriptor and its length are returned.

2 QAL_MAXMIN.C
Contains a suite of routines that determine the true minimum and
maximum latitude and longitude coordinates of ADRG data on a CD
from the ADRG QAL file (see the DMA ADRG product specification for
additional information about the QAL file).

QAL_MAXMIN.C contains two subroutines. In order to work
correctly, both routines must be invoked in the proper sequence.

3 QAL_MAXMIN_INIT
Initializes variables that are used by the routine QAL_MAXMIN_RET.
Hence, it must be invoked first. The following syntax is used to
invoke qal_maxmin_init:

        qal_maxmin_init (&scale, &zone)
where
        scale   contains the chart scale
                     (long, passed by reference)
        zone   contains the TS zone
                     (long, passed by reference)

3 QAL_MAXMIN_RET
Returns the minimum and maximum latitude and longitude values for
the data that have been determined from the QAL file on the CD.
This routine MUST be invoked last. The following syntax is used
to invoke qal_maxmin_ret:

qal_maxmin_ret (&minlon, &maxlon, &minlat, &maxlat)

where

minlon, maxlon	Contain minimum and maximum longitude coordinates (double, passed by reference)
minlat, maxlat	Contain minimum and maximum latitude coordinates (double, passed by reference)

3 Example

The following FORTRAN code segment provides an example of proper usage of QAL_MAXMIN.C:

```
C*** Variable declarations ****
            real*8    minlat, maxlat, minlon ,maxlon ! Contains
the
                                                !coordinates
        integer*4    zone/0/, scale/5/
        character*(80)   filename   ! Contains the full QAL file name

        filename = 'CDROM06:[ONA50101]ONA50101.QAL' // char(0)

C**** Note that FORTRAN, by default, ****
C**** passes arguments by reference   ****
        call qal_maxmin_init (scale, zone, %ref(filename))
        call qal_maxmin_ret  (minlon, maxlon, maxlat, minlat)
        print (6,*) minlon, maxlon, maxlat, minlat
```

2 QUAD_PROCS.C

A suite of C language routines that compute minimum and maximum latitude/longitude coordinates for polar map data. These functions are designed for use with polar data and will not effect nonpolar data. When the minimum and maximum latitude/longitude values of a polar region are constant (i.e., if this region was translated to a nonpolar representation, its shape would be rectangular), special measures must be taken to find the true minimum and maximum row and column values of the data contained within this region.

As a latitude/longitude point changes its latitude position in the polar region, it traces to an arc configuration. Conversely, as the point changes its longitude position, it traces a perfect circle. The point at which this circle crosses the polar quadrant axis (0, 90, -90, 180) is where the maximum or minimum row/column values lie. Thus, if a region is reduced to the quadrants it crosses, the true minimum and maximum row/column values can be computed.

These routines break a polar rectangular area into smaller areas that are bounded by quadrants (defined in terms of latitude/longitude). Subtopics include requirements, individual

F-93

routines within the suite, and an example of usage.

·3 Requirements
The following files must be included for use with QUAD_PROCS.C:

      V2_DIR:QUAD.H   Contains data definitions used by the
                          subroutines.

      V2_DIR:DATA_DEFS.H   Contains type definition for common
                          variables (e.g., scale, zone)

The following subroutines are required for linkage. For
additional information about each subroutine, see MDFFHELP under
the main topic CAC_SOURCE_CODE.

      LL_MAXMIN.C   A suite of routines that determine the true pair
                   of minimum and maximum latitude and longitude
                   coordinates.

      RC_MAXMIN.C   A suite of routines that determine the true pair
                   of minimum and maximum row/column pairs.

3 QUADRANT_LONGITUDE_BOUNDS
Passes in the starting longitude and the ending longitude values
of a region and should be called before functions
QUADRANT_LONGITUDE_RET and QUADRANT_LONGITUDE_NEXT are invoked.
The following syntax is used to invoke QUADRANT_LONGITUDE_BOUNDS:

      void quadrant_longitude_bounds (LON *left, LON *right)
where
     left   Contains a starting longitude coordinate (passed)
     right   Contains an ending longitude coordinate (passed)

3 QUADRANT_LONGITUDE_RET
When iteratively called until returning a FALSE value, this
routine returns all quadrants that lie within the initial
longitude bounds that were passed to QUADRANT_LONGITUDE_BOUNDS.
The following syntax is used to invoke QUADRANT_LONGITUDE_RET:

      BOOLEAN quadrant_longitude_ret (LON *left, LON *right)
where
     a BOOLEAN value returns TRUE if a new longitude
                            quadrant is found.
                          Otherwise, returns FALSE
     left   Contains a starting longitude coordinate
          (double, passed)
     right   Contains an ending longitude coordinate
          (double, passed)

3 QUADRANT_LONGITUDE_NEXT
Sets up the longitude values for the next call to

QUADRANT_LONGITUDE_RET. This function must be called last. The
following syntax is used to invoke QUADRANT_LONGITUDE_NEXT:

```
        void quadrant_longitude_next (LON *left, LON *right)
where
        left    Contains a starting longitude coordinate
                (double, passed)
        right   Contains an ending longitude coordinate
                (double, passed)
```

3 Example

The following code segment, which prints the coordinate values
-90, -180, 180, 90, provides an example of usage:

```
        left_lon  = -90;    /* stating longitude coordinate */
        right_lon =  90;    /* ending  longitude coordinate */

        /* Establish starting and ending longitude coordinates */
        quadrant_longitude_bounds (&left_lon, &right_lon);

        /* Obtain quadrants containing map data */

        while (quadrant_longitude_ret (&left_lon, &right_lon))
            {
             printf ("%lf %lf\n", left_lon, right_lon);
             /* Establish next coordinates to be used */
             quadrant_longitude_next (&left_lon, &right_lon);
            }
```

2 RC_MAXMIN.C

RC_MAXMIN.C, written in C language, contains routines that
determine the true pair of minimum and maximum row/column values
from a set of row/column pairs. RC_MAXMIN.C determines if the
row/column values transition 180 or 0 degrees and adjusts the
minimum and maximum row/column values accordingly.

3 Routines

RC_MAXMIN.C consists of three routines:

    RC_MAXMIN_INIT: Initializes variables used by the next two
                    calls. This routine MUST be called first.

    RC_MAXMIN_SEND: Repeat for each row/column pair within
                    the set.

    RC_MAXMIN_RET: The final routine and MUST be called last.
                    It returns the minimum and maximum row
                    and column values for all of the pairs passed
                    to RC_MAXMIN_SEND.

In order to work correctly, all three routines must be invoked in
proper sequence. The Example subtopic shows proper usage.

3 Example
  The following example uses pseudocode to show proper usage of
  RC_MAXMIN.C and its associated routines.

```
 call RC_MAXMIN_INIT ()   ! Initialize variables used by the
                          ! next two calls.  This routine MUST be
                          ! called first.

! Loop - repeat for each row/column pair within the set.

        LOOP 1 TO Number_Of_ROW_COLS

            ! Send in as many row/col values as you like.
            ! Row and Col are INTEGER*4 (LONG) and
            ! are passed by reference.

            call RC_MAXMIN_SEND (row, col)

        END LOOP

! This is the final routine and MUST be called last.  It returns
! the minimum and maximum row/column values from the set of
! row/column pairs passed to RC_MAXMIN_SEND.  Normal minimum and
! maximum values are used when the set has been determined to ONLY
! cross 0 degrees longitude.  However, when the set crosses 180, or
! both 0 and 180 degrees, the least positive and least negative
! column values are used for the maximum column and minimum column
! values, respectively.  Values are passed by reference

call RC_MAXMIN_RET (mincol, maxcol, minrow, maxrow)
```

2 SEND_MAIL.FOR
 Provides an interface to the VAX/VMS MAIL utility.
 This routine can only be invoked from VAX/VMS FORTRAN due to the
 passing of character strings using descriptors. The SEND_MAIL_C
 routine can be used to send VAX/VMS mail from a "C" language
 routine.

 No include files required for calling. However, this routine is
 VAX/VMS dependent.

3 Invocation
 The following syntax is used to invoke SEND_MAIL:

```
     call SEND_MAIL (send_to,subject,buffer,lines)
```

where

    send_to: VMS username to receive the mail message.
             (passed, character*(*))

    subject: Subject title.
             (passed, character*(*))

     buffer: Character string buffer containing the text of the
             mail message. This buffer may contain several lines of
             text.
             (passed, character*(*) array)

      lines: Number of lines contained in "buffer".
             (passed, integer*2)

Note: The variable "buffer" may be a single string; in the case of
a one line message or it may be a character array. For example,
if the declaration "character*80 message_buf(10)" is used with
"lines" equal to 10, SEND_MAIL will send a message of 10 lines
with 80 characters per line to the VMS username that is
specified in "send_to".

2 SEND_MAIL_C.C

Subroutine SEND_MAIL_C sends a VAX/VMS mail message from programs
written in C programming language. The subtopic Example shows
proper usage.

3 Example
The following pseudocode provides an example of SEND_MAIL_C
usage.

```
#include <stdio.h>
#include <string.h>

#define MAX_LINES_OF_MAIL  250   /* Max # of lines in message   */
#define MAX_LINE_LENGTH    250   /* Max # of characters in line */

main ()
 {

  char send_to[MAX_LINE_LENGTH]; /* Name of person(s) to send to */
  char subject[MAX_LINE_LENGTH]; /* Subject of message.          */

  char line_buffer[MAX_LINES_OF_MAIL][MAX_LINE_LENGTH];

  long lines;    /* Number of lines in message */

  strcpy (send_to, "SMITH, JONES");
  strcpy (subject, "TEST OF SEND_MAIL_C");
```

```
/* Insert Message */
strcpy (line_buffer[0], "                                    ");
strcpy (line_buffer[1], " HELLO,                             ");
strcpy (line_buffer[2], "                                    ");
strcpy (line_buffer[3], "This is a test of send_mail_c.c  ");
strcpy (line_buffer[4], " You can place up to 250 lines   ");
strcpy (line_buffer[5], " in your message and each line   ");
strcpy (line_buffer[6], " can be 250 characters in length.");
strcpy (line_buffer[7], "                                    ");

/* Number of lines in message */
lines = 9;

/* Send Mail */
send_mail_c (send_to, subject, line_buffer, lines);

}
```

2 TIMEM.MAR

This C language routine separates one quadword into two longwords
using the following protocol:

* is passed a VAX time string and its length

* builds a string descriptor

* calls BINTIM_S (a VAX system macro) which returns a quadword.
  The quadword contains the number of seconds (in 100
  nanoseconds) since 17-NOV-1858 00:00:00.00.

* converts this quadword into two (more manageable) longwords.

3 Example

The following example uses C-based pseudocode to show proper use
of TIMEM.MAR

```
    long one;      /* first long word, returned argument */
    long two;      /* second long word, returned */

    long length;   /* length of the character string TIMESTR */
                   /* passed argument                        */

                   /* VAX time string, passed argument       */
    char timestr[VAX_TIME_STR_LEN] = "12 AUG 1991 12:00:00";


    length = strlen (timestr);

    /* Get quadword and return as two longwords */
    TIMEM (&two, &one, timestr, length);
```

2 USE_ADRG_LOGICALS.FOR
Inserts the specified processing thread's logical name table into
the LNM$FILE_DEV logical name search path. The routine uses the
FORTRAN "common" block variables PROCESSING_CODE and PROCCODELEN
in the LOGICALS.INC file.

3 Invocation

The following syntax is used to invoke USE_ADRG_LOGICALS:

        call USE_ADRG_LOGICALS

No parameters are passed. This routine uses the FORTRAN "common"
block variables PROCESSING_CODE and PROCCODELEN that are located
in the LOGICALS.INC file.

3 Required_Logicals
It is assumed that the LNM$FILE_DEV logical name equivalence
before modification is as follows:
                        LNM$PROCESS
                        MDFF
                        LNM$IIS_FILE_DEV
                        LNM$JOB
                        LNM$GROUP
                        LNM$SYSTEM
                        DECW$LOGICAL_NAMES

This routine inserts the logical name table, specified by
PROCESSING_CODE and PROCCODELEN, before the LNM$PROCESS logical
name. This causes any processing thread definitions to be found
first. The include file LOGICALS.INC is required for invocation.

APPENDIX G
TOPIC FILE CNC_PROCESSING.HLP

1 CNC_Processing

This covers the processing of COMPRESSED NAUTICAL CHART (CNC) data.
Since both the CAC and the CNC have the same source data format,
ADRG, their processing steps are very similar.

See the CNC_Specifics topic for CNC specific information.

2 CNC_Specifics

```
        ------------ TO BE ADDED LATER ----------
        Put items that differ from CAC processing HERE.
```

The following text and subtopics appear when **Definitions** is selected as an **MDFFHELP** subtopic:

```
Definitions

 The following topics are definitions for terms used in the
 MDFF.

 Additional information available:

 Chart_Updating_Manual Color_Compression      Compression
 CORE_Segments            Datum      Dither_Pattern
 Downsample               EDGE_Segments
 Ellipsoid  FILLED_Segments      Logical_CDROM_Sector  Micron
 Neatlines  Pixel        Processing_Thread
 Row_and_Column_Coordinates       Segment    Spatial_Compression
 Template    TS_System_Coordinates       UNFILLED_Segments
 Vector_Quantization
```

The following text comprises this **MDFFHELP** topic file. Note that subtopics begin with the key "2", which is located in column 1.

1 Definitions
    The following topics are definitions for terms used in the
    MDFF.

2 Chart_Updating_Manual
    Chart Updating Manual (CHUM), the Defense Mapping Agency
    Aeronautical CHUM is a semiannual publication, with monthly
    supplements, that provides textual and/or graphic additions,
    deletions, or modifications of cartographic data to published
    aeronautical charts. Changes appearing in the CHUM are
    generally considered to be critical to flight safety.

2 Color_Compression
    The second phase of creating Compressed Aeronautical Charts,
    after transformation from ARC to Tesselated Spheroid, which
    reduces storage requirements by a factor of 3:1. Color
    compression is achieved by subjecting the image data to a
    vector quantization process that selects the closest match of
    240 entries in a color palette to represent each pixel.

2 Compression
    A reduction in the amount of space required to store a given
    set of data. The MDFF compresses Arc Digitized Raster Graphics
    to Compressed Aeronautical Chart by a factor of 48:1 in three

H-3

steps:

(1) Downsampling data from ARC to TS, which reduces resolution of data by four to one (from 256 pixels per inch to 128 pixels per inch).

(2) Color compressing data by three to one (from 24- to 8-bits per pixel).

(3) Spatially compressing data by four to one (from four 8-bit pixel groups to one 8-bit codeword).

2 CORE_Segments

A PASS1 processing term used to identify all full Tesselated Spheroid segments for a given ARC Digitized Raster Graphics CD-ROM.

2 Datum

There are two types of DATUM; horizontal and vertical. Both descriptions are provided.

Datum (horizontal)- The horizontal geodetic datum is uniquely defined by five quantities. Three of these, latitude(phi), longitude(lambda), and geoid height (N) are defined at the datum origin. The adoption of specific values for the geodetic latitude and longitude implies specific deflections of the vertical at the origin. A geodetic azimuth is often cited as a datum parameter, but the azimuth and longitude are precisely related by the Laplace condition so there is no need to define both. There are two other quantities that define the reference ellipsoid: the semimajor axis and flattening or the semimajor axis and semiminor axis.

Datum (vertical)- A level surface to which elevations are referred, usually mean sea level, but may also include mean low water, mean lower low water, or an arbitrary starting elevation(s).

2 Dither_Pattern

A matrix, utilizing fill patterns, that creates the illusion of intensity and color. Dither patterns vary from one another by how the matrix is filled.

2 Downsample

A term used to describe the process by which ARC Digitized Raster Graphics is transformed into Tesselated Spheroid. Downsampling produces a 4:1 reduction in the size of the data by decreasing data resolution from 256 to 128 pixels per inch (approximately, averages the values of four ARC Digitized Raster Graphics pixels to form one Tesselated Spheroid pixel).

2 EDGE_Segments

A PASS1 processing term used to identify all partially filled

Tesselated Spheroid segments for a given ARC Digitized Raster Graphics CD-ROM.

2 Ellipsoid
A surface whose plane sections (cross sections) are all ellipses or circles, or the solid enclosed by such a surface.

2 FILLED_Segments
A PASS3 processing term used to identify previous edge segments from PASS1, which have been merged with other edge segments to form a full segment.

2 Logical_CDROM_Sector
The smallest addressable block of data (2048 bytes) on an ISO 9660 CD-ROM.

2 Micron
A unit of measure; one millionth of a meter.

2 Neatlines
The lines that bound the body of a map, usually parallels and meridians, but may be conventional or arbitrary grid lines. Also called sheet lines.

2 Pixel
A pixel, or picture element, is the smallest entity in a raster graphic image.

2 Processing_Thread
A three letter code used to identify an MDFF processing build. Processing threads are only used internally within the MDFF. The first character distinguishes the data set type [(A)-CAC, (N)-CNC, (D)-DLMS, etc.]; the second letter identifies the scale of the data set [(0)-1:50K, (1)-1:100K, (2)-1:250K,etc.]; the third letter is a sequence identifier [(A)- first, (B)-second, etc.]

2 Row_and_Column_Coordinates
In Tesselated Spheroid, segments are arranged by latitudinally based rows and longitudinally based columns within the equatorial and temperate zones. Row 0 is located with its southern boundary on the equator. Positively numbered rows extend northward from row 0 in the northern hemisphere, and negatively numbered rows extend southward from row -1 in the southern hemisphere.

The first column of segments in each nonpolar zone (column 0) is located with its western boundary on the 0 degrees meridian. Positive columns extend eastward from column 0 and stop at the 180 degrees meridian. Negative columns extend westward from column 0 and stop at the 180 degrees meridian.

Within a single CAC segment, rows of pixels are numbered from bottom (row 0) to top (row 255). Columns of pixels are numbered from left (column 0) to right (column 255).

2 Segment
   An array of 256 rows with 256 pixels each, stored in band
   sequential form. Equivalent to approximately two square
   inches of paper chart.

2 Spatial_Compression
   The third phase of processing CACs, which reduces storage
   requirements by a factor of 4:1. Spatial compression
   is achieved by applying a vector quantization process that
   first classifies the image and then replaces each 2 by 2
   pixel block in the image with a coded entry from a 256-entry
   lookup table.

2 Template
   A definition file applied here to MDFF mapstations, used to
   define a coverage of a given area at a given scale. The
   template may then be used to transfer data from a CAC to
   an aircraft optical disc.

2 TS_System_Coordinates
   In the nonpolar zones, Tesselated Spheroid coordinates are the
   WGS 84 phi and lambda under the equirectangular projection. In
   the polar zones, Tesselated Spheroid coordinates are created by
   rotating the polar coordinates to the equatorial zone, in the
   equirectangular projection.

2 UNFILLED_Segments
   A PASS3 processing term used to identify previous edge segments
   from PASS1 which, after merging, are still unfilled.

2 Vector_Quantization
   Vector_Quantization, used by the MDFF applies the "lossy"
   compression method to both color and spatially compress ARC
   Digitized Raster Graphics data. Uses a training set of data to
   predict colors and pixel patterns over a larger set of data.

APPENDIX I
TOPIC FILE DLMS_PROCESSING.HLP

1 DLMS_Processing

This topic will cover the processing of Digital Land Mass System (DLMS) data.

--------------- TO BE ADDED LATER -----------

I-3

APPENDIX J
TOPIC FILE HINTS.HLP

TOPIC FILE: HINTS.HLP

The following text and subtopics appear when **Hints** is selected as
an **MDFFHELP** subtopic:

```
HINTS

Type the name of one of the categories listed below to obtain
a list of related commands and topics.  To obtain detailed
information on a topic press RETURN until you reach the
"Topic?" prompt and then type the name of the topic.

 Additional information available:

 Executing_Programs      Program_Development     Logical_Names
 Contacting_People       Directories_and_Files
 Batch_and_Print_Jobs
```

The following text comprises this **MDFFHELP** topic file. Note that
subtopics begin with the key "2", which is located in column 1.

1 Hints
 Type the name of one of the categories listed below to obtain a
 list of related commands and topics. To obtain detailed
 information on a topic press RETURN until you reach the "Topic?"
 prompt and then type the name of the topic.

2 Executing_Programs

 The NRL Map Data Formatting Facility utilizes a variety of
 programs that are available for execution. A pseudostandard
 naming convention has been adopted to facilitate finding the
 appropriate program. This standard requires a program name to be
 descriptive such that it includes the program's purpose and use.
 For example, DISPLAY_CAC_VWS is a program that displays CAC data
 on a Vax Workstation (VWS) monitor.

 In general, programs that require the use of a specific monitor
 should include the monitor as part of the program name. For
 example, DISPLAY_CAC_IVAS is a program which displays CAC data on
 an IVAS monitor. Programs that display data should have the word
 DISPLAY in the title.

 In order to execute programs, the user must have a processing
 thread assigned to their process. Processing threads are assigned
 using the following symbol:

$ ADRGLOGS ###

where ### is the desired processing thread name. Almost all of the
programs for the MDFF require that a processing thread be
assigned.   For more information on processing threads see the
topic Processing_Threads under HINTS.

All of the programs for the MDFF should reside in the directory
MDFFEXE:. A directory listing of this area will show the names of
the executable programs that are available.

In order to execute a specific program the following command
should be typed at the $ prompt.

        $ RUN/NODEB MDFFEXE:Program_Name

For more information on specific programs look under the main
topic CAC_Program_Descriptions. This topic contains functional
descriptions of programs used by the MDFF.

2 Program_Development
  Source code for the majority of programs that have been developed
  at the MDFF is stored in the V2_DIR (i.e., Software Version 2)
  directory. Programs within the V2_DIR directory should be used as
  a guide for future program development. Because of their modular
  nature, some of the subprograms may be utilized as part of new
  software development.

  Files that have an extension of .FOR are written in the FORTRAN
  programming language, and those with an extension of .C are
  written in the C programming language.

  In addition to source code that resides in the V2_DIR directory,
  there is a library of routines that is also available for
  inclusion in programs. This library, named MDFFLIB.OLB, is linked
  using the same method as any other VAX/VMS object library. This
  following command will list names of the object modules that are
  contained within the library:

        $ LIB/LIST MDFFLIB.OLB

  See MDFFHELP main topics CAC_Program_Descriptions for program
  descriptions and CAC_SOURCE_CODE for subprogram descriptions.

2 Logical_Names
  Logical names are used extensively by the MDFF for various
  purposes. A logical name is a string that represents another,
  usually longer, string. The main benefit of using logical names
  is that they are typically used as descriptive strings for
  physical devices. For example, the logical name CHART_SEGS serves
  as a descriptive name for the disk storage device DUA12:. Logical
  names are maintained by VMS and may be translated by using the

                         J-4

following command.

        $SHOW LOGICAL abcd

Where abcd is a logical name.   For additional information on
logical names see the VAX/VMS help.

For additional information on specific logical names see the main
MDFFHELP topic Logical_Names.

2 Contacting_People
The MDFF has several methods for contacting people via VAX mail.
For information about VAX mail, see the VMS HELP topic MAIL.
There are three distribution lists that can be used for sending
messages to a certain group of people.

MDFF_PROCESSORS.DIS   Sending a message using this list will
                             contact people who are only involved in
                             the CAC processing.

LOCAL_MDFF.DIS   Sending a message using this list will
                             contact all people who are directly
                             involved in the MDFF.

GLOBAL_MDFF.DIS   Sending a message using this list will
                             contact everyone having an account
                             on one of the MDFF computers.

These distribution lists are located in the MDFFEXE directory.

2 Directories_and_Files
The MDFF uses many different directories and files in the
production of a CAC. The directory containing source code and
executable images that are used by the MDFF to produce a CAC is
named MDFFEXE.

CAC processing involves several formats of chart data (e.g., ADRG,
downsampled, and compressed) and other associated files (e.g.,
command procedures and log files).

The following is a list of major directories and the types of
files are contained therein.

[MDFF.XXX.SCRATCH]

Contains all of the command procedures and log files for
downsampling and compressing ADRG data. This directory is known
as "MDFF_SCRATCH" for a specific processing thread (as noted by
"XXX") and is defined by the logical name MDFF_SCRATCH. There are
five subdirectories that exist under this directory, descriptions
of each follow:

[.CB_REPAIR]   Contains all the segments that need to be
                 repaired. Each segment is in a subdirectory
                 for the row that segment is in.

[.CDYXXXXX]   Contains the SEGMENTS.DAT file of downsampled
                 segments for the YXXXXX CD-ROM. Note that "Y"
                 designates the Mastering Facility code and
                 "XXXXX" designates the CAC CD-ROM number.

[.CODEBOOKS]   Contains all the codebook build procedures
                 that have been used for this CAC.

[.VALIDATE]   Contains a copy of the transmittal header
                 file, TRANSH01.THF, that is used in the CD-ROM
                 validation procedure.

[.PAXXXXX]   Contains all of the PA (palette) codebook
                 compression procedures.

CHART_SEGS:[MDFF.XXX.CHART_SEGMENTS.]

This is known as the "CHART_SEGMENTS" directory and it contains
row subdirectories for every row that is contained within area of
the XXX chart ODI build. Each row subdirectory contains the
downsampled segment files for that particular row. The logical
name CHART_SEGS serves as a descriptive name for the disk storage
device.

CHART_ODI:[MDFF.XXX]

The chart ODI directory, contains the structure of the image that
will be written onto CAC CD-ROM. The logical name CHART_ODI
serves as a descriptive name for the disk storage device. One
subdirectory exists:

[.MAPX]   The root of the tree structure for a CAC. The
                 chart scale is denoted by the "X". The following
                 subdirectories are associated:

[.CDYXXXXX]   This directory and subdirectory tree contains
                 all of the header information for that
                 particular CD-ROM. Note that "Y" designates the
                 Mastering Facility code and "XXXXX" designates
                 the CAC CD-ROM number.

[.PAXXXXXX]   This directory contains the COVERAGE.DAT file
                 for that particular PA (i.e., palette), the row
                 directories containing all the compressed
                 segment files, and the color PALETTE.DAT file

J-6

2 Batch_and_Print_Jobs

To facilitate an easy data flow through the process of a chart ODI build, the MDFF uses batch queues to manage the scheduling for the reading of ADRG data, downsampling, compression, and print jobs.

The primary queue is named MDFF_BATCH. This queue manages the reading of ADRG CD-ROMs, downsampling, and the building of command procedures. MDFF_BATCH manages every type of activity except for the submission of codebook compression jobs, which are handled by a thread-specific batch queue.

Thread-specific batch queues parcel out codebook compression jobs to one or more generic codebook queues. Each generic codebook queue executes on a separate computer within the MDFF cluster. There are seven generic codebook queues for compression jobs and, depending on the priority of a certain thread, the number of generic queues that are utilized by one particular thread may vary.

TOPIC FILE: LOGICAL_NAME.HLP

1 Logical_Names

Much of the MDFF functionality is achieved through the use of logical names. Most MDFF logical names are simply descriptive abbreviations of longer strings. For example, V2_DIR is an MDFF logical name that represents the directory name DUA12:[MDFF.SOFTWARE.NOARL.SRC.V2].

Hence, the logical name V2_DIR represents the directory where all the version 2 MDFF software is located. The MDFF uses logical names for almost every aspect of the V2 software including directory designations and queue names.

Use the following command to show current definitions for all existing logical name tables:

> $ SHOW LOGICAL *

The primary use of logical names is to facilitate easier multi-thread processing of CACs. For example, in order to access data (i.e., downsampled data, compressed data, etc.) that have been processed as part of thread A4A, all of the logical names associated with A4A must be set. This is done by using the following symbol:

> $ ADRGLOGS A4A

Use the following command to show current definitions for all logical names associated with processing thread A4A:

> $ SHOW LOGICAL/TABLE=A4A

K-3

APPENDIX L
TOPIC FILE MAPSTATION.HLP

The following text and subtopics appear when **Mapstation** is selected as an **MDFFHELP** subtopic:

```
Mapstation

Four mapstations are used in the MDFF to create Aircraft Optical
Disks (AODs) that contain user-selected areas and scales of CAC
data.  NRL's mapstations are 386-based  PCs that have been
specifically designed by Horizons Technology Inc. (HTI) to
permit an operator to select predetermined areas and scales of
CAC data and output this data onto militarized AODs, which are
write-once, read-many (WORM) devices.

This topic provides descriptions of processing procedures.
Familiarity with mapstations and procedures is assumed.

 Additional information available:

 Logging_CAC_CDROMS      Template_Creation      ODI_Transfer
 Save_ODI_to_MO          Miscellaneous          Paper_Charts
```

The following text comprises this **MDFFHELP** topic file. Note that subtopics begin with the key "2", that is located in column 1.

1 Mapstation

Four mapstations are used in the MDFF to create Aircraft Optical
Disks (AODs) that contain user-selected areas and scales of CAC
data. NRL's mapstations are 386-based PCs that have been
specifically designed by Horizons Technology Inc. (HTI) to
permit an operator to select predetermined areas and scales of
CAC data and output this data onto militarized AODs, which are
write-once, read-many (WORM) devices.

This topic provides descriptions of processing procedures.
Familiarity with mapstations and procedures is assumed.

2 Logging_CAC_CDROMS

The following procedure is used to add new CAC CD-ROMS into the
mapstation database.

A - Select the "IMPORT MEDIA" option from the main menu.

B - Select the "CD-ROM" option from the "IMPORT MEDIA" menu.

C - Insert a CAC CD-ROM into the CD-ROM drive (device J:) and
    press the RETURN key.

D - Upon completion, note the addition in the appropriate
    logbook.

2 Template_Creation

Templates are used to define the data being transferred and must
be created in order to perform an ODI transfer. Templates include
the area of coverage and data characteristics (e.g., scale). The
following procedure (steps A through K) is used to create a
template:

A - Select "DEFINE ODI" option from the main menu.

    NOTE: The coverage area of template definition must be logged

          into the mapstation database (see topic
          Logging_CAC_CDROMS).

B - Select the "NEW ODI" option from the "DEFINE ODI" menu.

C - Select the appropriate scale (1:500K is the default scale).

D - The world map will be displayed on the screen. Select the
    region of the world from which the template will be created
    by scrolling to the desired location and pressing <RETURN>.

E - The regional map will be displayed. Scroll to the desired
    area, which will be included in the new template, and press
    <RETURN>. The limits of available coverage will be outlined
    in red. The new template must reside within this area of
    coverage.

F - Select the "ADD COVERAGE" option from the "DEFINE ODI" menu.

G - Move the arrow on the display to a corner of the area to be
    defined by the template and press <RETURN>. H - Scroll the
    arrow until the desired coverage for the template
    is enclosed by a box and press <RETURN>. If more areas need
    to be defined, repeat steps G and H. When finished defining
    an ODI, press <ESC>.

    NOTE: If coverage is to be deleted: select the "DELETE
    COVERAGE" option from the "DEFINE ODI" menu, place the
    arrow within the box to be deleted and press <RETURN>.

I - Select the "ESTIMATE SIZE" option to determine the size of
    the template coverage.

    NOTE: The maximum size of an ODI subdirectory is 80 sectors.
    The maximum number of subdirectories per zone per scale is
    three.

L-4

J - Select the "VIEW AREA" option from the "DEFINE ODI" menu and view data that are within the template area for the defined scale and zone(s). If colors are incorrect, verify that the proper palettes (*.PAL and *.AOD) are on the fixed disk.

K - Once steps I and J are completed, select the "SAVE ODI" option from the "DEFINE ODI" menu and name the template.

2 ODI_Transfer

In order to perform ODI transfers, the palettes for template coverage must be located on the fixed disk and in directory C:\PALETTES

The following procedure (steps A through F) is used to transfer an ODI.

A - Select the "TRANSFER ODI" option from the main menu.

B - Select output media from the "TRANSFER ODI" menu: the "AIRCRAFT DISK" option for WORM media , or the "OPSTA ODI" option for MO media.

NOTE: An ODI image file can also be written to the fixed disk

by selecting "AIRCRAFT DISK" option and leaving the DMU turned off. This enables individual ODI builds to be saved, merged, and transferred (after quality assurance is performed

on the image on fixed disk) before writing a WORM (see the topic Save_ODI_to_MO)

C - If writing onto WORM media, clean the disk before mounting it

into the DMU.

NOTE: Make sure write-protection is off. Initially, write protection is on.

D - If writing onto WORM media, turn DMU power on.

E - Select the template to be transferred.

F - Insert any media that is prompted by the software (such as a specific CAC CD-ROM).

2 Save_ODI_to_MO

Optical Disk Image (ODI) generation can require a significant amount of time and effort. Therefore, once an ODI has been created, it is usually saved by transferring it from fixed disk (drive E:) onto Magneto-Optical (MO) disk.

L-5

The following procedure (steps A through K) is used to merge individual ODI builds to a single ODI and then copy that ODI onto an Aircraft Optical Disk (AOD).

A - Turn on the mapstation and change to the default current directory on the fixed disk (drive E:) by typing E:

B - Put the MO into the F: drive and type DIR F: to display the list of ODI filenames.

C - Delete any old ODI files and ODIHEAD.TEX files that are on the E: drive.

D - Type LOCK.

E - Copy an ODI from drive F: to E:ACOD.ODI
    The file name ACOD is derived from Aircraft Optical Disk.

F - Type UNLOCK.

G - Construct an ODI image from individual ODI builds by executing the program C:\HTI_UTILS\ADDSUB xx. This program will merge subdirectory xx from the ACOD.ODI file on E: drive

    into an ACOD.OUT file.

H - Repeat steps D through G until all ODI files on F: have been

    merged.

I - Upon completion of the ODI image build, rename the file ACOD.OUT to ACOD.ODI. Make a backup copy of ACOD.ODI onto a MO with the following DOS command:

        COPY E:ACOD.ODI F:filename.ODI

    where filename is descriptive of the geographic area.

J - Retain header information that is contained in the EMPTY template by using the DOS copy command to copy the file C:\HTI_UTILS\EMPTY.TEX to the file E:ODIHEAD.TEX.

K - Put the WORM into the Digital Memory Unit (DMU), making sure that it is secured.

L - Change directory to the C: drive, and type MAP to invoke the mapstation software.

M - Select the "TRANSFER ODI" option and then select the "TEMPLATE TO AOD" option.

N - Use the EMPTY template by pressing the ENTER key.

O - The copy will begin.

2 Miscellaneous

The following topics contain informational notes.

3 Drives_Devices
The following drives and devices are available on the
mapstations.

Drive   Device
 A: Three and a half inch floppy drive
 B: Three and a half inch floppy drive
 C: Hard disk drive, partitioned with D:
 D: Hard disk drive, partitioned with C:
 E: Hard disk drive
 F: MO drive
 J: CDROM drive

NOTE:
Drives C: and D: are based on a single partitioned hard disk
drive. These drives contain system and mapstation software.

Drive E: contains output ODI files, which by mapstation
software, will be transferred onto WORM media.

3 Palettes
Original mapstation palettes, provided by HTI, are different
from the NRL palettes. On NRL's mapstations (mapstations
2,3,6 and 7), NRL palettes have replaced HTI palettes for the
JNC,TPC, and JOG scales in all zones.

HTI naming conventions for mapstation palettes also differ from
NRL conventions. HTI mapstation palettes use scale values 1
through 6 where 1 = 1:2M ... 6 = 1:50K. NRL CAC uses scale
values 0 through 5 where 0=1:50K and 5=1:2M.

Palettes are located in the directory C:\PALETTES. Palette
names use the format PAsz.PAL where "s" indicates scale and "z"
indicates zone.

Color tables for each palette use the same prefix but have
an .AOD suffix. Hence, color tables use the format PAsz.AOD
where "s" indicates scale and "z" indicates zone. The
mapstation color table is pixel interleaved (red green blue,
red green blue, red green blue...) whereas NRL's color tables
are band interleaved (256 red, 256 green, 256 blue). To copy
NRL palettes to a mapstation, use the program REPL_PAL

L-7

(available on mapstation 3, C: drive).

3 View_Area
   Before transferring ODI data to an AOD, verify that the
   mapstation has the proper palette for the defined transfer area.

   Use the VIEW AREA option from the DEFINE ODI menu to identify
   the transfer palette. Viewing data from the VIEW MEDIA menu can
   be deceiving because the palette used for viewing from this menu
   is NOT the palette transferred to the AOD!

3 Size_Requirements
   The true maximum size per subdirectory is 80 sectors (this is
   less than the mapstation ODI TRANSFER limit). Because a sector
   contains approximately 1 MB, the maximum subdirectory size on a
   WORM is approximately 80 MB.

   The total amount of data that can fit onto WORM media is 260 MB.
   However, the size of the ODI file on fixed disk may be larger.

2 Paper_Charts
   Paper charts may be scanned, processed and added to the
   mapstation database.

3 Scanning

   JOG paper charts normally require about 12 scans each. The
   chart must be folded and laid face down in the upper-left
   corner of the scanner (view from front) in a horizontal
   direction. The folds in the chart should be as straight as
   possible with the scan containing at least nine easily definable

   points (control points for warping).

   Occasionally, a dark piece of paper may need to be placed
   over the portion of chart being scanned because colors from
   the reverse side of the chart may be seen in the scan.

   After all the scans for a chart have been completed, copy the
   scanned files onto a MO (using the DOS copy command). The
   scanned files are located in scale-specific directories. For
   example, when using JOG data, scanned files are located in
   directory D:/1-250K/. After three or so charts, some of these
   scanned image files must be deleted from the fixed disk to make
   room for additional scanned image files.

3 Processing
   The following procedure (steps A through K) is used to process
   scanned charts.

   A - Using scanned image files from a MO, use the following DOC
       command to copy all files to drive and directory D:/1-250K/

COPY F:*.SC D:/1-250K/*.*

B - Type MAP to invoke the mapstation software.

C - Select the "PROCESS CHARTS" option, followed by the
    "SELECT IMAGE" option, and chose one of the scanned
    images that has just been copied.

D - Select the "CONTROL POINTS option followed by the
    "ADD/MODIFY" option to add the control points to the
    image. Usually, GEODETIC system coordinates are the
    easiest to use, unless the image does not contain three
    longitudinal lines. In the case of having only two
    longitudinal lines, it may be easier to use the Universal
    Transverse Mercator (UTM) system. The UTM system is
    selected by pressing the space bar.

E - If the GEODETIC system is selected, enter nine sets of
    latitude/longitude coordinates. These coordinates are used
    as control points. If the UTM system is selected, refer
    to the paper chart to obtain the required nine UTM
    coordinates. A blue box is located at the bottom of the
    paper chart that indicates now to use UTM coordinate
    system.

F - For the last selection (coordinates) look for the blue
    two-letter symbol (inclusive of the points you want to
    select). The number will first be read vertically then
    it will be read horizontally. For example, if the number
    "9" is on the vertical line and "3" is on the horizontal
    line, on which you're picking your control point within the
    box ST, the resulting coordinate would be ST 900300.

G - Once all nine control points have been selected and
    ALIGNED, select the "PROCESS IMAGE" option. This will
    warp the image and take about three to five minutes to
    complete. Save the warped image. Repeat steps C through H

    until all scanned images have been warped.

H - Exit the mapstation software. Save the warped image files
    by using the following DOS command to copy files *.WP1 onto
    MO media (which is contained on F: drive):

        COPY *.WP1 F:*.*

    Warped images are stored in the same directory as scanned
    images.

NOTE: It is extremely important to perform the save operation
      before scanned image processing is continued. The warped
      files are needed to log the data files into the other

mapstations.

I - Once the warped images have been saved, invoke the
    mapstation software by typing, at the DOS level, MAP.
    Select the "PROCESS CHARTS" option, then select the "LOG
    IMAGES" option. This step will log the scanned image files
    into the workstation database. After this step has
    completed, warped images are automatically deleted from the
    disk.

J - The last step involves reviewing the logged in scanned
    data. This is done by going into the "DEFINE ODI" menu,
    and selecting the "NEW ODI" option, the appropriate scale,
    and the "VIEW AREA" option. This review will alert you to
    any gaps, missing segments, or serious alignment problems
    that may have occurred. Use this same procedure to
    perform appropriate repairs.

APPENDIX M
TOPIC FILE PROCESSING_THREAD.HLP

1 Processing_Threads

Processing threads allow the MDFF to process more than one CAC at
a time. This enables greater flexibility and a higher CAC
production rate. The names of processing threads are descriptive;
all thread names are three characters in length with each
character having the following meanings:

   Example: A4A

      A: Type of charts being compressed
                 A - Aeronautical
                 N - Nautical
                 T - Test

      4:    Scale of the charts being compressed
                 0 - TLM   (1:50K scale,   TLM50)
                 1 - XXX   (1:100K scale, TLM100)
                 2 - JOG   (1:250K scale)
                 3 - TPC   (1:500K scale)
                 4 - ONC   (1:M scale)
                 5 - JNC   (2:M scale)
                 etc.

      A: Is used to delineate simultaneous chart ODI builds,
            when both charts are at the same scale.

1 Symbols
A symbol defines a symbolic name for a character string or
integer value. The MDFF uses symbols as descriptive abbreviations
for VMS commands. MDFF symbols are declared globally; hence, they
are available to all users. The following command may be used to
list existing symbols (note, user-defined symbols will also be
included).

      $ SHOW SYMBOL *

Examples of MDFF symbols and their translated meaning include:

   MOUNT_CDROM translates to the command @MDFFEXE:MOUNT_CDROM
      ADRGLOGS translates to the command @MDFFEXE:USE_ADRG_LOGICALS

M-3

APPENDIX N
TOPIC FILE SYMBOLS.HLP

1 Symbols
A symbol defines a symbolic name for a character string or
integer value. The MDFF uses symbols as descriptive abbreviations
for VMS commands. MDFF symbols are declared globally; hence, they
are available to all users. The following command may be used to
list existing symbols (note, user-defined symbols will also be
included).

        $ SHOW SYMBOL *

Examples of MDFF symbols and their translated meaning include:

    MOUNT_CDROM translates to the command @MDFFEXE:MOUNT_CDROM
        ADRGLOGS translates to the command @MDFFEXE:USE_ADRG_LOGICALS